



Intel[®] IXP2400/IXP2800 Network Processor

Programmer's Reference Manual

November 2003

Order Number: 278746-014



Revision History

Revision Date	Revision	Description
07/12/01	001	Internal release.
09/07/01	002	Internal release. Updated instructions
10/10/01	003	Release for Customer Information Book V0.3.
10/31/01	004	Pre-Release V 1.
11/04/01	005	Update for Customer Information Book V0.3
01/25/02	006	IXA SDK 3.0 Pre-Release II
04/22/02	007	Release for the IXA SDK 3.0
08/02/02	008	First combined IXP2400/IXP2800 version for the IXA SDK 3.0 Pre-Release 4
11/01/02	009	Release for the IXA SDK 3.0 Pre-Release 5
01/22/03	010	Release for the IXA SDK 3.0 Pre-Release 6
05/31/03	011	Release for the IXA SDK 3.1 Pre-Release 2
07/3/03	012	Release for the IXA SDK 3.1 Pre-Release 3
09/8/03	013	Release for the IXA SDK 3.5
11/16/03	014	Release for the IXA SDK 3.5 Pre-Release 2

Information in this document is provided in connection with Intel® products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The IXP2400/IXP2800 Network Processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license. The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document. Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the express written consent of Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at <http://www.intel.com>.

Copyright © Intel Corporation, 2003

Intel and Intel XScale are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries

*Other names and brands may be claimed as the property of others.

Contents

1	Introduction.....	21
	1.1 About this Document.....	21
	1.2 Related Documentation.....	21
2	Assembler	23
	2.1 Acronyms	23
	2.2 Definitions.....	23
	2.3 Source File Elements	24
	2.3.1 Instructions	24
	2.3.2 Directives.....	24
	2.3.3 Comments	24
	2.4 Block Structure	25
	2.5 Assembly Process Steps.....	25
	2.6 Assembler Preprocessor	26
	2.6.1 Preprocessor Reserved Labels	27
	2.6.2 Preprocessor Operation	27
	2.6.3 Constant Expressions (const-expr)	28
	2.6.3.1 Preprocessor Binary & Unary Operators	29
	2.6.3.2 Preprocessor: Functions	29
	2.6.3.3 STRING Operator.....	30
	2.6.3.4 LOG2() Function.....	31
	2.6.3.5 Preprocessor Function Examples	32
	2.6.4 Macros and Expansion Token Restriction.....	33
	2.6.5 Syntax for Argument and Token lists	33
	2.6.6 Leading and Trailing Spaces in Macros	34
	2.6.7 Environment Variables	34
	2.6.8 Predefined Processor Type and Revision Symbols	34
	2.6.9 Predefined Import Variables.....	38
	2.7 Preprocessor Usage Techniques.....	38
	2.7.1 Branching into a Macro	38
	2.7.2 Constructing Names from Numbers	40
	2.8 Registers and Signals	41
	2.8.1 Register Naming Conventions.....	42
	2.8.1.1 Indexed Registers	42
	2.8.1.2 Mixing Indexed and Named Register Usage.....	46
	2.8.1.3 Transfer Registers (xfer)	46
	2.8.2 Register Declarations	46
	2.8.2.1 Preferred Register Declaration Syntax.....	47
	2.8.2.2 Details of Volatile and Visible	51
	2.8.2.3 Compatible Register Declaration Syntax.....	51
	2.8.2.4 Dealing with self-write neighbor regs	52
	2.8.3 Aggregate and Array Support.....	52
	2.8.3.1 Register Arrays.....	52
	2.8.3.2 Compatibility with Earlier Releases	53
	2.8.3.3 Doubled Signal References.....	54
	2.8.3.4 Usage Notes.....	55
	2.8.3.5 Compatibility Issues	56

2.8.4	Transfer Order(.xfer_order).....	56
2.8.5	Register Lifetime Details	57
2.8.5.1	MEv2 Queue Information	58
2.8.6	Signal Declarations	58
2.8.7	Use of REMOTE Keyword.....	59
2.8.8	Address Operator	60
2.8.8.1	Accumulating Results for ctx_arb[--]	62
2.8.8.2	Examples of Address Operator and Visible/Volatile Signals	63
2.8.9	Signal Lifetime Details.....	64
2.8.10	Register Allocation Directives	64
2.8.11	GPR A/B Bank Conflicts.....	69
2.8.11.1	Automatic A/B Bank Conflict Resolution	70
2.8.12	GPR Spilling.....	70
2.8.13	Lifetime Out-Of-Register Errors.....	71
2.8.13.1	Transfer Register Lifetimes	74
2.9	Assembler Optimizer.....	75
2.10	Assembler Directives	77
2.10.1	Summary of Directives	77
2.11	Directives Definitions.....	79
2.11.1	Token Replacement (#define, #undef).....	79
2.11.2	Optimization Directives.....	80
2.11.3	Loops	81
2.11.3.1	For Loops (#for, #endloop).....	81
2.11.3.2	Repeat Loops (#repeat, #endloop)	81
2.11.3.3	While Loops (#while, #endloop)	82
2.11.4	Macros (#macro, #endm)	82
2.11.5	Conditional Assembly (#ifdef, #if, #else, #elif, #endif)	84
2.11.6	Error Reporting (#error).....	85
2.11.7	File Inclusion (#include)	86
2.11.8	Import Variable (.import_var).....	86
2.11.9	Code block directive (.begin, .end).....	87
2.11.10	Manual Register Allocation (.addr).....	88
2.11.11	Memory Allocation Directives.....	89
2.11.12	Memory Block and Register Initialization	90
2.11.13	Local Memory Mode Directives	91
2.11.14	Number of Contexts Directive	91
2.11.15	Initial Next Neighbor Mode Directive.....	91
2.11.16	Operand Synonym (.operand_synonym)	91
2.11.17	Structured Assembly	92
2.11.17.1	Conditional (.if, .elif, .else, .endif, if_unsigned, .elif_unsigned).....	92
2.11.17.2	Repeat Loops (.repeat, .until).....	93
2.11.17.3	While Loops (.while, .endw)	93
2.11.17.4	Break and Continue	93
2.11.17.5	Conditional Expressions.....	93
2.11.17.6	Errors	95
2.11.18	Structured Assembly Usage Considerations.....	96
2.11.19	Warning Directives	97
2.12	Subroutine Definition (.subroutine, .endsub).....	98
2.13	Linker Directives.....	98

3	MEv2 Instruction Set	101
3.1	Instruction Syntax.....	103
3.1.1	Restricted and Unrestricted Src and Dest Operands	103
3.1.1.1	Two Source Operand Selection Rules	104
3.1.2	I/O Instruction Format.....	104
3.1.2.1	Source Operands (src_op1, src_op2)	104
3.1.2.2	Reference Count (ref_cnt).....	105
3.1.2.3	Optional Tokens (opt_tok)	105
3.1.2.4	Event Signals	108
3.1.3	Condition Codes.....	111
3.1.4	Branch Defer (defer[n]).....	112
3.1.5	Coding Restrictions	113
3.1.5.1	Branch or I/O Command in Defer Slot.....	113
3.1.5.2	Condition Codes after Swap.....	114
3.1.5.3	CAM after Conditional P3 Branch	114
3.1.5.4	Dram with Swap	115
3.1.5.5	BCC after Conditional P3 branch	115
3.1.5.6	LOCAL_CSR_RD cannot be in last defer slot.....	117
3.1.5.7	LOCAL_CSR_WR to ACTIVE_LM_ADDR, or CAM_LOOKUP	117
3.1.5.8	LOCAL_CSR_RD must be followed by an IMMED op	118
3.1.5.9	I/O Command Op after LOCAL_CSR_WR.....	119
3.1.5.10	LOCAL_CSR_WR to CTX_WAKEUP_EVENTS	120
3.1.6	MEv2 Permitted Coding Sequences	121
3.1.6.1	Swap after P3 Branch	121
3.1.6.2	Memory Command after P3 Branch.....	121
3.1.6.3	Swap after Voluntary Swap.....	122
3.1.6.4	A LOCAL_CSR_WR in defer slot.....	122
3.1.6.5	LOCAL_CSR_WR can be followed by a LOCAL_CSR_RD or LOCAL_CSR_WR.....	123
3.2	Instruction Set	124
3.2.1	ALU	124
3.2.2	ALU_SHF	126
3.2.3	ASR	128
3.2.4	BCC (BRANCH CONDITION CODE).....	129
3.2.5	BR	130
3.2.6	BR_BCLR, BR_BSET	131
3.2.7	BR=BYTE, BR!=BYTE	132
3.2.8	BR=CTX, BR!=CTX.....	133
3.2.9	BR_INP_STATE, BR_!INP_STATE	134
3.2.10	BR_SIGNAL, BR_!SIGNAL	135
3.2.11	BYTE_ALIGN_BE, BYTE_ALIGN_LE	136
3.2.12	CAM_CLEAR	139
3.2.13	CAM_LOOKUP	140
3.2.14	CAM_READ_TAG	142
3.2.15	CAM_READ_STATE	143
3.2.16	CAM_WRITE	144
3.2.17	CAM_WRITE_STATE	145
3.2.18	CAP (Enumerated CSR Addressing)	146
3.2.19	CAP (Calculated Addressing).....	148
3.2.20	CAP (Reflect)	153

3.2.21	CRC_LE, CRC_BE	155
3.2.22	CTX_ARB.....	158
3.2.23	DBL_SHF	160
3.2.24	DRAM (Read and Write)	161
3.2.25	DRAM (RBUF and TBUF)	163
3.2.26	FFS	165
3.2.27	HALT	166
3.2.28	HASH	167
3.2.29	IMMED	170
3.2.30	IMMED_B0, IMMEDIATE_B1, IMMEDIATE_B2, IMMEDIATE_B3.....	172
3.2.31	IMMED_W0, IMMEDIATE_W1.....	173
3.2.32	JUMP	174
3.2.33	LD_FIELD, LD_FIELD_W_CLR	175
3.2.34	LOAD_ADDR	176
3.2.35	LOCAL_CSR_RD.....	177
3.2.36	LOCAL_CSR_WR	178
3.2.37	MSF (Media Switch Fabric).....	179
3.2.38	MUL_STEP	182
3.2.39	NOP	184
3.2.40	PCI	185
3.2.41	POP_COUNT	187
3.2.42	RTN.....	188
3.2.43	SCRATCH (Read & Write)	189
3.2.44	SCRATCH (Atomic Operations).....	191
3.2.45	SCRATCH (Ring Operations).....	193
3.2.46	SRAM (Read & Write)	195
3.2.47	SRAM (Atomic Operations).....	197
3.2.48	SRAM (CSR).....	201
3.2.49	SRAM (Read Queue Descriptor).....	203
3.2.50	SRAM (Write Queue Descriptor).....	207
3.2.51	SRAM (Enqueue)	209
3.2.52	SRAM (Dequeue).....	213
3.2.53	SRAM (Ring Operations).....	216
3.2.54	SRAM (Journal Operations)	218
4	Address Maps	221
4.1	Intel XScale™ Address Map.....	221
4.1.1	DRAM Memory and Intel XScale™ Core Flash ROM (2GB).....	222
4.1.2	SRAM Memory (1GB)	222
4.1.3	CAP-CSRs (32MB).....	223
4.1.3.1	ME Transfer and Local CSRs.....	224
4.1.3.2	Peripherals	225
4.1.3.3	CAP CSRs.....	225
4.1.4	SlowPort - Flash ROM (64M)	226
4.1.5	MSF (32M)	226
4.1.6	Scratch (32M).....	227
4.1.7	SRAM CSRs and Queue Array (64MB)	228
4.1.8	DRAM CSRs (32M)	229
4.1.9	Intel XScale™ Core Local CSRs (32M).....	230
4.1.9.1	Hash Operations	230

4.1.10	PCI IO (32M)	231
4.1.11	PCI CFG (32M)	231
4.1.12	PCI Special Cycles / IACK (32M)	232
4.1.13	PCI Configuration Registers (32M)	232
4.1.14	PCI Controller CSRs	232
4.1.15	PCI Memory (1/2GB).....	232
4.2	PCI Address Map	233
4.2.1	DRAM Memory Space.....	234
4.2.2	SRAM Memory Space.....	235
4.2.3	CSR Memory Space.....	236
4.3	Microengine Address Map.....	238
5	Control and Status Registers (CSRs).....	241
5.1	Introduction.....	241
5.1.1	IXP2800 and IXP2400 CSR Summary	241
5.1.2	Register Notation Conventions.....	242
5.1.3	Reserved Fields	242
5.2	Microengine Local CSRs.....	243
5.2.1	USTORE_ADDRESS	247
5.2.2	USTORE_DATA_LOWER, USTORE_DATA_UPPER.....	248
5.2.3	USTORE_ERROR_STATUS	249
5.2.4	ALU_OUT	249
5.2.5	TIMESTAMP_HIGH, TIMESTAMP_LOW	250
5.2.6	ACTIVE_CTX_FUTURE_COUNT	250
5.2.7	INDIRECT_CTX_FUTURE_COUNT	251
5.2.8	ACTIVE_FUTURE_COUNT_SIGNAL.....	251
5.2.9	INDIRECT_FUTURE_COUNT_SIGNAL.....	251
5.2.10	PROFILE_COUNT	252
5.2.11	PSEUDO_RANDOM_NUMBER.....	253
5.2.12	NEXT_NEIGHBOR_SIGNAL	253
5.2.13	PREV_NEIGHBOR_SIGNAL	254
5.2.14	SAME_ME_SIGNAL.....	255
5.2.15	ACTIVE_CTX_STS	255
5.2.16	INDIRECT_CTX_STS	256
5.2.17	CTX_ARB_CNTL	257
5.2.18	CTX_ENABLES.....	257
5.2.19	CC_ENABLE	260
5.2.20	CSR_CTX_POINTER.....	260
5.2.21	ACTIVE_CTX_SIG_EVENTS.....	261
5.2.22	INDIRECT_CTX_SIG_EVENTS.....	261
5.2.23	ACTIVE_CTX_WAKEUP_EVENTS	261
5.2.24	INDIRECT_CTX_WAKEUP_EVENTS	261
5.2.25	ACTIVE_LM_ADDR_0	262
5.2.26	ACTIVE_LM_ADDR_1	262
5.2.27	INDIRECT_LM_ADDR_0	262
5.2.28	INDIRECT_LM_ADDR_1	262
5.2.29	BYTE_INDEX	263
5.2.30	T_INDEX	264
5.2.31	T_INDEX_BYTE_INDEX.....	264
5.2.32	INDIRECT_LM_ADDR_0_BYTE_INDEX.....	265

	5.2.33	INDIRECT_LM_ADDR_1_BYTE_INDEX.....	265
	5.2.34	ACTIVE_LM_ADDR_0_BYTE_INDEX.....	265
	5.2.35	ACTIVE_LM_ADDR_1_BYTE_INDEX.....	265
	5.2.36	NN_PUT.....	265
	5.2.37	NN_GET.....	266
	5.2.38	CRC_REMAINDER.....	267
	5.2.39	LOCAL_CSR_STATUS.....	267
5.3		RDR DRAM Controller - IXP2800.....	268
	5.3.1	RDRAM_CONTROL (# = 0,1,2).....	269
	5.3.2	RDRAM_ERROR_STATUS_1 (# = 0,1,2).....	271
	5.3.3	RDRAM_ERROR_STATUS_2 (# = 0,1,2).....	272
	5.3.4	RDRAM_ECC_TEST (# = 0,1,2).....	273
	5.3.5	RDRAM_SERIAL_COMMAND (# = 0,1,2).....	274
	5.3.6	RDRAM_SERIAL_DATA (# = 0,1,2).....	275
	5.3.7	RDRAM_CONFIG_1 (# = 0,1,2).....	275
	5.3.8	RDRAM_CONFIG_2 (# = 0,1,2).....	277
	5.3.9	RDRAM_CONFIG_3 (# = 0,1,2).....	278
	5.3.10	RDRAM_RAC_INIT (# = 0,1,2).....	279
	5.3.11	RDRAM_MISC_RAC_CONTROL.....	281
	5.3.12	RDRAM_RAC_CONFIG.....	281
	5.3.13	RDRAM_1066_CONFIG_GROUP (# = 0,1,2).....	282
	5.3.14	RDRAM_SERIAL_CONFIG (# = 0,1,2).....	282
	5.3.15	RDRAM_K0 through RDRAM_K11 (# = 0,1,2).....	283
5.4		DDR SDRAM Controller - IXP2400.....	285
	5.4.1	DDR SDRAM Register Map.....	285
	5.4.2	DRAM Controller Control Register (DU_CONTROL).....	285
	5.4.3	DRAM Error Status Register 1 (DU_ERROR_STATUS_1).....	288
	5.4.4	DRAM Error Status Register 2 (DU_ERROR_STATUS_2).....	289
	5.4.5	DRAM ECC Test Register (DU_ECC_TEST).....	290
	5.4.6	DRAM Initialization Register (DU_INIT).....	292
	5.4.7	DRAM Controller Control Register 2 (DU_CONTROL2).....	293
	5.4.8	DRAM RCOMP & I/O Registers.....	294
		5.4.8.1 DDR_Rx_DLL.....	298
		5.4.8.2 DDR_Rx_Deskew.....	299
		5.4.8.3 DDR_RDDLSEL_RECVEN.....	299
5.5		SRAM QDR Controller.....	300
	5.5.1	SRAM_CONTROL.....	302
	5.5.2	SRAM_PARITY_STATUS_1.....	304
	5.5.3	SRAM_PARITY_STATUS_2.....	304
	5.5.4	SPARE.....	305
	5.5.5	QDR_INTERNAL_PIPELINE.....	306
	5.5.6	QDR_RX_DLL.....	306
	5.5.7	QDR_RX_DESKEW.....	307
	5.5.8	QDR_RD_PTR_OFFSET.....	307
	5.5.9	QDR RCOMP Registers.....	308
		5.5.9.1 Q_RCMP_SETUP_CONTROL.....	308
		5.5.9.2 Q_RCMP_PMOS_MEASURED.....	311
		5.5.9.3 Q_RCMP_NMOS_MEASURED.....	311
		5.5.9.4 Q_RCMP_PMOS_OVERRIDE.....	312
		5.5.9.5 Q_RCMP_NMOS_OVERRIDE.....	312

5.5.9.6	Q_RCMP_PMOS_NMOS_SCOMP_OVERRIDE (IXP2400 and IXP2800 Rev A).....	313
5.5.9.7	Q_RCMP_PMOS_NMOS_SCOMP_OVERRIDE(IXP2800 Rev B)	314
5.5.9.8	Q_RCMP_STRENGTH_SLEW_INDEX_SEL	315
5.5.9.9	Q_RCMP_ADDR_PMOS_PU_OFFSET	317
5.5.9.10	Q_RCMP_ADDR_NMOS_PD_OFFSET	317
5.5.9.11	Q_RCMP_DATA_PMOS_PU_OFFSET	317
5.5.9.12	Q_RCMP_DATA_NMOS_PD_OFFSET	318
5.5.9.13	Q_RCMP_K_CLK_PMOS_PU_OFFSET	318
5.5.9.14	Q_RCMP_KCLK_NMOS_PD_OFFSET	318
5.5.9.15	Q_RCMP_DQ_PMOS_PU_OFFSET	319
5.5.9.16	Q_RCMP_DQ_NMOS_PD_OFFSET	319
5.5.9.17	Q_RCMP_PMOS_NMOS_VERT_OVERRIDE	319
5.5.9.18	Slew Rate Tables	320
5.5.10	QDR unit initialization	322
5.5.10.1	IXP2800 A Steppings QDR initial setup procedure	322
5.5.10.2	IXP2800 B Steppings - QDR initial setup procedure	324
5.5.10.3	IXP2400 QDR initial setup procedure	326
5.6	CSR Access Proxy (CAP)	327
5.6.1	Scratchpad Memory CSRs (CAP CSR)	327
5.6.1.1	SCRATCH_RING_BASE_# (# = 0 -15).....	328
5.6.1.2	SCRATCH_RING_HEAD_# (# = 0 - 15)	329
5.6.1.3	SCRATCH_RING_TAIL_# (#= 0 - 15).....	330
5.6.2	Hash Configuration (CAP CSR)	330
5.6.2.1	HASH_MULTIPLIER_48_# (# = 0,1).....	331
5.6.2.2	HASH_MULTIPLIER_64_# (# = 0,1).....	331
5.6.2.3	HASH_MULTIPLIER_128_# (# = 0,1,2,3).....	332
5.6.3	Fast Write CSRs (CAP CSR)	333
5.6.3.1	THD_MSG (Generic).....	334
5.6.3.2	THD_MSG_CLR_#_\$_& (# = {0,1}, \$= {0,7 or 3}, & = {0,7})... 335	
5.6.3.3	THD_MSG_#_\$_& (# = {0,1}, \$ = {0,7 or 3}, & = {0,7})	335
5.6.3.4	THD_MSG_SUMMARY_#_\$_ (# = {0,1}, \$ = {0,1})	336
5.6.3.5	SELF_DESTRUCT_# (# = 0 -1)	336
5.6.3.6	INTERTHREAD_SIG.....	337
5.6.3.7	XSCALE_INT_# (# = A, B)	337
5.6.4	Global Control (CAP CSR)	338
5.6.4.1	PRODUCT_ID	338
5.6.4.2	MISC_CONTROL	339
5.6.4.3	MSF Clock Control CSR (MCCR) - IXP2400 only	340
5.6.4.4	IXP_RESET_0	344
5.6.4.5	IXP_RESET_1	347
5.6.4.6	CLOCK_CONTROL	348
5.6.4.7	STRAP_OPTIONS	350
5.6.4.8	WATCHDOG_HISTORY	351
5.6.5	Timer (CAP CSR).....	352
5.6.5.1	T#_CTL (# = 1,2,3,4).....	352
5.6.5.2	T#_CLD, (# = 1,2,3,4).....	353
5.6.5.3	T#_CSR, (# = 1,2,3,4)	354
5.6.5.4	T#_CLR(# = 1,2,3,4).....	354
5.6.5.5	TWDE	354
5.6.6	GPIO (CAP CSR)	355
5.6.6.1	GPIO_PLR	356

	5.6.6.2	GPIO_PDPR	357
	5.6.6.3	GPIO_PDSR	357
	5.6.6.4	GPIO_PDCR	358
	5.6.6.5	GPIO_POPR	358
	5.6.6.6	GPIO_POSR, GPIO_POCR	358
	5.6.6.7	GPIO_REDR, GPIO_FEDR	359
	5.6.6.8	GPIO_EDSR	360
	5.6.6.9	GPIO_LSHR, GPIO_LSLR	361
	5.6.6.10	GPIO_LDSR	362
	5.6.6.11	GPIO_INER	362
	5.6.6.12	GPIO_INSR	362
	5.6.6.13	GPIO_INCR	363
	5.6.6.14	GPIO_INST	363
5.6.7		UART (CAP CSR)	364
	5.6.7.1	UART_RBR	365
	5.6.7.2	UART_THR	365
	5.6.7.3	UART_DLRL, UART_DLRH	365
	5.6.7.4	UART_IER	366
	5.6.7.5	UART_IIR	367
	5.6.7.6	UART_FCR	369
	5.6.7.7	UART_LCR	371
	5.6.7.8	UART_LSR	373
	5.6.7.9	UART_SPR	375
5.6.8		PMU (Performance Monitor UNit) (CAP CSR)	376
	5.6.8.1	PMUCONTCFG—PMU Control Bus Configuration Register ..	378
	5.6.8.2	PMUSTAT—PMU Counter Interrupt Status Registers	380
	5.6.8.3	PMUMASK—PMU Counters Interrupt Mask Registers	382
	5.6.8.4	PMUINTEN—PMU Interrupt Enable Register	386
	5.6.8.5	CHAPCMDN—CHAP Command N Register (N = 0..5)	387
	5.6.8.6	CHAPEVN—CHAP Events N Register (N = 0..5)	391
	5.6.8.7	CHAPSTAT# (# = 0..5)	393
	5.6.8.8	CHAPDATAN—CHAP Data N Register (N = 0..5)	394
5.6.9		SlowPort (CAP CSR)	395
	5.6.9.1	SP_CCR	396
	5.6.9.2	SP_WTC1	398
	5.6.9.3	SP_WTC2	399
	5.6.9.4	SP_RTC1	401
	5.6.9.5	SP_RTC2	401
	5.6.9.6	SP_FSR	403
	5.6.9.7	SP_PCR	404
	5.6.9.8	SP_ADC	404
	5.6.9.9	SP_FAC	405
	5.6.9.10	SP_FRM	405
	5.6.9.11	SP_FIN	406
	5.6.9.12	SP_TXE	406
	5.6.9.13	SP_RXE	407
5.7		Media and Switch Fabric Interface (MSF) - IXP2800	408
	5.7.1	MSF_RX_CONTROL	413
	5.7.2	MSF_TX_CONTROL	418
	5.7.3	MSF_INTERRUPT_STATUS	421
	5.7.4	MSF_INTERRUPT_ENABLE	424
	5.7.5	CSIX_TYPE_MAP	425
	5.7.6	FC_EGRESS_STATUS	425

5.7.7	FC_INGRESS_STATUS	427
5.7.8	FC_STATUS_OVERRIDE.....	429
5.7.9	MSF_CLOCK_CONTROL.....	430
5.7.10	FCIFIFO	432
5.7.11	FCEFIFO	433
5.7.12	RX_DESKEW_# (# = pin name).....	433
5.7.13	SPI4_DYNFILT_THRESH.....	434
5.7.14	MSF_DLL_DATA_DELAY_CTL	435
5.7.15	FC_DYNFILT_THRESH.....	436
5.7.16	FC_DLL_DATA_DELAY_CTL.....	437
5.7.17	HWM_CONTROL.....	438
5.7.18	RX_THREAD_FREELIST_# (# = 0,1,2).....	439
5.7.19	RX_PORT_MAP.....	441
5.7.20	RBUF_ELEMENT_DONE	441
5.7.21	RX_CALENDAR_LENGTH	441
5.7.22	FCEFIFO_VALIDATE.....	442
5.7.23	TX_SEQUENCE_# (# = 0,1,2)	442
5.7.24	RX_THREAD_FREELIST_TIMEOUT_# (# = 0,1,2).....	443
5.7.25	RX_PORT_CALENDAR_STATUS_# (0 TO 255)	443
5.7.26	TX_CALENDAR_LENGTH.....	444
5.7.27	TX_CALENDAR_# (# = 0 - 255).....	445
5.7.28	TX_PORT_STATUS_# (# = 0 - 255).....	445
5.7.29	TX_MULTIPLE_PORT_STATUS_# (# = 0 - 15)	445
5.7.30	TBUF_ELEMENT_CONTROL_\$_# (\$ = A, B, # = Element No)	446
5.7.31	TRAIN_DATA	449
5.7.32	TRAIN_CALENDAR	452
5.7.33	TRAIN_FLOW_CONTROL.....	453
5.7.34	RX_PHASEMON_# (# = pin name).....	455
5.7.35	MSF_IO_BUF_CTL	457
5.7.36	FC_IO_BUF_CTL.....	458
5.7.37	MSF Initial Setup Procedure for the IX2800 Rev A	459
5.7.38	MSF Initial Setup Procedure for the IX2800 Rev B	460
5.8	Media and Switch Fabric Interface(MSF) - IXP2400	462
5.8.1	IXP2400 MSF Address Map.....	462
5.8.2	MSF_Rx_Control.....	464
5.8.3	MSF_Tx_Control	466
5.8.4	MSF_Interrupt_Status	470
5.8.5	MSF_Interrupt_Enable	472
5.8.6	CSIX_Type_Map.....	473
5.8.7	FC_Egress_Status	473
5.8.8	FC_Ingress_Status.....	474
5.8.9	HWM_CONTROL.....	475
5.8.10	SRB_Override	478
5.8.11	Rx_Thread_Freelist_{0..3}	479
5.8.12	RBUF_Element_Done	480
5.8.13	Rx_MPHY_Poll_Limit	480
5.8.14	FCEFIFO_Validate	481
5.8.15	Rx_Thread_Freelist_Timeout_{0..3}	482
5.8.16	Tx_Sequence_{0..3}.....	482
5.8.17	Tx_MPHY_Poll_Limit	483

5.8.18	Tx_MPHY_Status.....	484
5.8.19	Tx_MPHY_Status_Extension.....	487
5.8.20	Rx_UP_Control_{0..3}.....	489
5.8.21	Tx_UP_Control_{0..3}.....	492
5.8.22	Rx_FIFO_Control_{0,1,2,3}.....	493
5.8.23	MSF_Rx_RCOMP_Status.....	495
5.8.24	MSF_Tx_RCOMP_Status.....	495
5.8.25	MSF_Rx_RCOMP_Override.....	496
5.8.26	MSF_Tx_RCOMP_Override.....	497
5.8.27	FCIFIFO.....	497
5.8.28	FCEFIFO.....	498
5.8.29	TBUF_ELEMENT_CONTROL_\$_# (\$= A, B, # = Element No).....	498
5.9	PCI.....	502
5.9.1	PCI Configuration Space.....	502
5.9.1.1	PCI_VEN_DEV_ID.....	503
5.9.1.2	PCI_CMD_STAT.....	503
5.9.1.3	PCI_REV_CLASS.....	504
5.9.1.4	PCI_CACHE_LAT_HDR_BIST.....	505
5.9.1.5	PCI_CSR_BAR.....	505
5.9.1.6	PCI_SRAM_BAR.....	506
5.9.1.7	PCI_DRAM_BAR.....	507
5.9.1.8	PCI_SUBSYS.....	507
5.9.1.9	PCI_INT_LAT.....	508
5.9.1.10	PCI_RCOMP_OVERRIDE.....	508
5.9.1.11	PCI_RCOMP_STATUS (IXP2400 Rev A and IXP2800).....	509
5.9.1.12	PCI_RCOMP_STATUS (IXP2400 Rev B).....	511
5.9.1.13	PCI_IXP_PARAM.....	511
5.9.2	PCI Controller CSRs.....	513
5.9.2.1	PCI_OUT_INT_STATUS.....	515
5.9.2.2	PCI_OUT_INT_MASK.....	515
5.9.2.3	MAILBOX_#.....	516
5.9.2.4	XSCALE_DOORBELL.....	516
5.9.2.5	XSCALE_DOORBELL_SETUP.....	517
5.9.2.6	PCI_DOORBELL.....	517
5.9.2.7	PCI_DOORBELL_SETUP.....	518
5.9.2.8	CHAN_#_BYTE_COUNT.....	518
5.9.2.9	CHAN_#_PCI_ADDR.....	519
5.9.2.10	CHAN_#_DRAM_BAR.....	519
5.9.2.11	CHAN_#_DESC_PTR.....	520
5.9.2.12	CHAN_#_CONTROL.....	520
5.9.2.13	CHAN_#_ME_PARAM.....	523
5.9.2.14	DMA_INF_MODE.....	523
5.9.2.15	PCI_SRAM_BAR_MASK.....	524
5.9.2.16	PCI_DRAM_BAR_MASK.....	525
5.9.2.17	PCI_CONTROL.....	526
5.9.2.18	PCI_ADDR_EXT.....	531
5.9.2.19	XSCALE_ERR_STATUS.....	531
5.9.2.20	XSCALE_ERR_ENABLE.....	534
5.9.2.21	XSCALE_INT_STATUS.....	536
5.9.2.22	XSCALE_INT_ENABLE.....	538
5.9.2.23	ME_PUSH_STATUS.....	539
5.9.2.24	ME_PUSH_ENABLE.....	539
5.10	Intel XScale® Core Local CSRs.....	541

5.10.1	Interrupt Controller (Intel XScale™ Core).....	541
5.10.1.1	{IRQ,FIQ}RAW_STATUS.....	545
5.10.1.2	{IRQ,FIQ}STATUS.....	547
5.10.1.3	{IRQ,FIQ}ENABLE.....	547
5.10.1.4	{IRQ,FIQ}ENABLE_SET.....	547
5.10.1.5	{IRQ,FIQ}ENABLE_CLR.....	548
5.10.1.6	{IRQ,FIQ}SOFT_INT.....	548
5.10.1.7	SCRATCH_RING_STATUS.....	548
5.10.1.8	{IRQ,FIQ}ERR_RAW_STATUS.....	549
5.10.1.9	{IRQ,FIQ}ERR_STATUS.....	550
5.10.1.10	{IRQ,FIQ}ERR_ENABLE.....	551
5.10.1.11	{IRQ,FIQ}ERR_ENABLE_SET.....	551
5.10.1.12	{IRQ,FIQ}ERR_ENABLE_CLR.....	551
5.10.1.13	{IRQ,FIQ}RAW_ATTN_STATUS.....	552
5.10.1.14	{IRQ,FIQ}ATTN_STATUS.....	552
5.10.1.15	{IRQ,FIQ}ATTN_ENABLE.....	553
5.10.1.16	{IRQ,FIQ}ATTN_ENABLE_SET.....	553
5.10.1.17	{IRQ,FIQ}ATTN_ENABLE_CLR.....	554
5.10.1.18	{IRQ,FIQ}THD_RAW_STATUS_#_# (\$= A, B and # = 0 - 3).....	554
5.10.1.19	{IRQ,FIQ}THD_STATUS_#_# (\$= A, B and # = 0 - 3).....	555
5.10.1.20	{IRQ,FIQ}THD_ENABLE_#_# (\$= A, B and # = 0 - 3).....	555
5.10.1.21	{IRQ,FIQ}THD_ENABLE_SET_#_# (\$= A, B and # = 0 - 3).....	556
5.10.1.22	{IRQ,FIQ}THD_ENABLE_CLR_#_# (\$= A, B and # = 0 - 3).....	556
5.10.2	Hash Operation (Intel XScale™ Core).....	556
5.10.2.1	HASH_OP_48_# (# = 0,1).....	557
5.10.2.2	HASH_OP_64_# (# = 0,1).....	558
5.10.2.3	HASH_OP_128_# (# = 0,1,2,3).....	558
5.10.2.4	HASH_DONE.....	559
5.10.3	Breakpoint (Intel XScale™ Core).....	560
5.10.3.1	BRK_RAW_STATUS.....	562
5.10.3.2	BRK_STATUS.....	563
5.10.3.3	BRK_ENABLE.....	563
5.10.3.4	BRK_ENABLE_SET.....	563
5.10.3.5	BRK_ENABLE_CLR.....	564
5.11	Intel XScale™ Co-Processors.....	564
5.12	MSF differences between IXP2400 and IXP2800.....	564
A	UCA Warnings.....	567
A.1	Introduction.....	567
A.2	UCA Warning (level 4) 4101.....	569
A.3	UCA Warning (level 1) 4700.....	569
A.4	UCA Warning (level 3) 4701.....	569
A.5	UCA Warning (level 2) 4702.....	570
A.6	UCA Warning (level 1) 5000.....	571
A.7	UCA Warning (level 3) 5002.....	571
A.8	UCA Warning (level 1) 5003.....	571
A.9	UCA Warning (level 3) 5004.....	572
A.10	UCA Warning (level 1) 5007.....	572
A.11	UCA Warning (level 4) 5008.....	573
A.12	UCA Warning (level 1) 5009.....	574
A.13	UCA Warning (level 1) 5011.....	574
A.14	UCA Warning (level 3) 5012.....	575
A.15	UCA Warning (level 2) 5100.....	575

A.16	UCA Warning (level 2) 5101	576
A.17	UCA Warning (level 2) 5102	576
A.18	UCA Warning (level 2) 5103	576
A.19	UCA Warning (level 2) 5104	577
A.20	UCA Warning (level 1) 5114	577
A.21	UCA Warning (level 1) 5115	577
A.22	UCA Warning (level 1) 5116	578
A.23	UCA Warning (level 2) 5117	578
A.24	UCA Warning (level 2) 5118	578
A.25	UCA Warning (level 3) 5121	579
A.26	UCA Warning (level 4) 5122	579
A.27	UCA Warning (level 4) 5124	580
A.28	UCA Warning (level 4) 5125	580
A.29	UCA Warning (level 4) 5126	580
A.30	UCA Warning (level 4) 5127	581
A.31	UCA Warning (level 4) 5128	581
A.32	UCA Warning (level 2) 5129	582
A.33	UCA Warning (level 2) 5130	582
A.34	UCA Warning (level 1) 5131	582
A.35	UCA Warning (level 2) 5132	583
A.36	UCA Warning (level 3) 5133	583
A.37	UCA Warning (level 4) 5134	583
A.38	UCA Warning (level 1) 5135	584
A.39	UCA Warning (level 1) 5136	584
A.40	UCA Warning (level 1) 5137	584
A.41	UCA Warning (level 1) 5138	585
A.42	UCA Warning (level 1) 5139	585
A.43	UCA Warning (level 1) 5140	585
A.44	UCA Warning (level 1) 5141	586
A.45	UCA Warning (level 1) 5142	586
A.46	UCA Warning (level 1) 5143	587
A.47	UCA Warning (level 3) 5144	587
A.48	UCA Warning (level 1) 5145	588
A.49	UCA Warning (level 1) 5146	588
A.50	UCA Warning (level 1) 5147	589
A.51	UCA Warning (level 2) 5148	589
A.52	UCA Warning (level 2) 5149	590
A.53	UCA Warning (level 1) 5150	590
A.54	UCA Warning (level 4) 5151	591

Figures

2-1	Assembly Process.....	26
2-2	Processor Type Constant Values.....	36
2-3	Bank Allocation Diagram.....	69
2-4	Example of a .lvr File.....	72
2-5	Example of a .lvr File Without a Real uword	72
2-6	Lifetime Register Spreadsheet.....	73

3-1	Load Immediate.....	170
3-2	Read Queue Descriptor Commands	205
3-3	Write Queue Descriptor Commands	208
3-4	Enqueue One Buffer at a Time using the Enqueue Command	211
3-5	Enqueue a String of Buffers to a Queue	212
3-6	Dequeue Buffer	214
3-7	Example of the Three Dequeue Modes.....	215
4-1	Four GB (32 bit) Intel XScale™ Address Space Divided among Various Targets	221
4-2	Four GB (32 bit) PCI Address Space	234
5-1	Conceptual Diagram of Counter Array	377
5-2	Count Types Example	392
5-3	Breakpoint Implementation.....	561

Tables

2-1	Acronym Definitions	23
2-2	Summary of Preprocessor Directives.....	27
2-3	Binary & Unary Operators	29
2-4	Functions.....	29
2-5	Examples of log2() Function.....	32
2-6	Processor Type Symbols	35
2-7	Revision Symbols.....	35
2-8	Predefined Import Variables.....	38
2-9	Registers Used By Contexts in Context-Relative Addressing Mode	45
2-10	MEv2 Logical Queues	58
2-11	Assembler Directives.....	77
2-12	Optimization List for #pragma optimize Directive	80
2-13	Condition directives	84
2-14	Error Reporting Severity Levels	86
2-15	Register Mapping - Context Relative to Absolute.....	88
2-16	pos and const Values	95
2-17	Linker Directives.....	98
3-1	Summary of Microengine Instructions	101
3-2	Source/Destination Choices for Addressing Modes	103
3-3	Legal Combinations of Source Operands	104
3-4	Reference Count Sizes	105
3-5	I/O Command Token Descriptions	105
3-6	Instructions and Optional Tokens that use Signals	108
3-7	Signal Restrictions for each I/O Instruction [command]	109
3-8	Branch Defer Summary.....	112
3-9	Branch on Condition Code Instructions.....	129
3-10	Initial Register Contents	136
3-11	Initial Register Contents	137
3-12	CAM_LOOKUP Result	141
3-13	CAM_READ_STATE Result	143
3-14	Enumerated CAP CSR Registers.....	146
3-15	CAP Indirect Format (Read and Write Commands).....	147

3-16	CAP Field Definitions	147
3-17	CAP Bit Map Address Field Encoding (src_op1 + src_op2).....	150
3-18	CAP Calculated Address Field Encoding (src_op1 + src_op2).....	151
3-19	CAP Indirect Format (Read and Write Commands).....	151
3-20	CAP Field Definitions	151
3-21	CAP (Reflect) Indirect Format	154
3-22	CAP (Reflect) Field Definitions.....	154
3-23	DRAM Indirect Format	161
3-24	DRAM Field Definitions	161
3-25	DRAM RBUF_RD & TBUF_WR Indirect Format.....	163
3-26	DRAM RBUF_RD & TBUF_WR Field Definitions	163
3-27	Number of S-Transfer Registers Used by Hash Instruction	167
3-28	Hash Indirect Format.....	168
3-29	Hash Field Definitions	168
3-30	Data Format in Transfer Registers.....	168
3-31	MSF Indirect Format	180
3-32	MSF Field Definitions	180
3-33	RBUF / TBUF Offset Address 128 64-Byte Elements	181
3-34	RBUF / TBUF Offset Address 64 128-Byte Elements	181
3-35	RBUF / TBUF Offset Address 32 256-Byte Elements	181
3-36	PCI Address Space.....	185
3-37	PCI Indirect Format	186
3-38	PCI Field Definitions.....	186
3-39	Scratch (Read and Write) Indirect Format	189
3-40	Scratch (Read and Write) Indirect Field Definitions	189
3-41	Scratch (Atomic Operations) Indirect Format.....	192
3-42	Scratch (Atomic Operations) Indirect Field Definitions.....	192
3-43	SCRATCH Ring Number Encoding (src_op1 + sr_op2)	193
3-44	SCRATCH Ring Indirect Format	194
3-45	SCRATCH Ring Indirect Field Definitions	194
3-46	SRAM (Read and Write) Indirect Format	195
3-47	SRAM (Read and Write) Indirect Field Definitions	195
3-48	SRAM Indirect Format (IXP28xx Rev A: all Atomics; IXP28xx Rev B: Pull Atomics)	199
3-49	SRAM Indirect Field Definitions (IXP28xx Rev A: all Atomics; IXP28xx Rev B: Pull Atomics)	199
3-50	SRAM Indirect Format (IXP28xx Rev B: no_pull Atomics).....	200
3-51	SRAM Indirect Field Definitions (IXP28xx Rev B: no_pull Atomics)	200
3-52	SRAM CSR Indirect Format	201
3-53	SRAM CSR Field Definitions.....	202
3-54	SRAM (Read Queue Descriptor) Indirect Format	206
3-55	SRAM (Read Queue Descriptor) Field Definitions	206
3-56	SRAM (Enqueue) Indirect Format.....	210
3-57	SRAM (Enqueue) Field Definitions	210
3-58	SRAM (dequeue) Indirect Format	214
3-59	SRAM (dequeue) Field Definitions.....	214
3-60	SRAM Ring Descriptor Format.....	216
3-61	SRAM Ring Size Encoding	216
3-62	SRAM Ring Indirect Format	217
3-63	SRAM Ring Indirect Field Definitions	217

3-64	SRAM Journal Indirect Format	220
3-65	SRAM Journal Indirect Field Definitions	220
4-1	Flash ROM - DRAM Mapping	222
4-2	SRAM Address Map for the Intel XScale™ Core	223
4-3	ME Transfer register and Local CSR Address Map for the Intel XScale™ Core	224
4-4	ME Transfer Register Addresses	225
4-5	Peripherals Address Map for the Intel XScale™ Core	225
4-6	CAP CSR Address Map for the Intel XScale™ Core	225
4-7	Slow Port Address Map for the Intel XScale™ Core	226
4-8	MSF Address Map for the Intel XScale™ Core	226
4-9	RBUF/ TBUF Offset Address 128 64-Byte Elements	227
4-10	RBUF/ TBUF Offset Address 64 128-Byte Elements	227
4-11	RBUF/ TBUF Offset Address 32 256-Byte Elements	227
4-12	Scratch Address Map	228
4-13	SRAM Queue Array Address for the Intel XScale™ Core	229
4-14	DRAM CSRs	229
4-15	Intel XScale™ Core CSRs	230
4-16	Intel XScale™ Core Hash Operand and Results Registers	230
4-17	PCI I/O Space	231
4-18	PCI Configuration Space	231
4-19	PCI Configuration Space	232
4-20	IXP2400/IXP2800 PCI Configuration Space	232
4-21	IXP2400/IXP2800 PCI Controller CSR Space	232
4-22	IXP2400/IXP2800 PCI Configuration Space	233
4-23	PCI Address Offset vs SRAM Controller	235
4-24	CSR Memory Space for PCI	236
4-25	CAP CSR Memory Space Breakdown for PCI	237
4-26	ME I/O Access	238
5-1	CSR Summary	241
5-2	Register Notation Conventions	242
5-3	Microengine Local CSR Summary	243
5-4	Microengine Local CSR Latencies	245
5-5	NN_PUT Ring Behavior	266
5-6	NN_PUT Ring Latency	266
5-7	RDR DRAM Register Summary	268
5-8	Address Bank Remapping (Optimize RDRAMs)	270
5-9	Address Bank Remapping (Optimize Banks)	271
5-10	RDRAM Constants (Hexadecimal) for 3-Channel Mode Part 1	283
5-11	RDRAM Constants (Hexadecimal) for 3-Channel Mode Part 2	284
5-12	RDRAM Constants (Hexadecimal) for 3-Channel Mode, Part 3	284
5-13	DDR SDRAM Register Map	285
5-14	RR_SYND values and error bit position mapping	290
5-15	DRAM RCOMP & I/O Configuration Register Map	294
5-16	SRAM Register Summary	300
5-17	Queueing Modes	304
5-18	Strength Control Settings	315
5-19	SRAM Register Summary (where # = 0,1,2,3)	320
5-20	Slew Table Format: IXP2400	321
5-21	Slew Table Format: IXP2800	321
5-22	Slew Rate Table Recommended Initial Values (IXP2800)	321

5-23	Slew Rate Table Recommended Initial Values (IXP2400)	322
5-24	Scratchpad Memory Register Summary	327
5-25	Head/Tail Use and Full Threshold by Ring Size	329
5-26	Hash Multiplier Register Summary	330
5-27	Inter-Process Communication Register Summary	333
5-28	Global Chassis Registers	338
5-29	Timer Register Map	352
5-30	GPIO Register Map	355
5-31	UART Register Map	364
5-32	Interrupt Conditions	367
5-33	Interrupt Identification Register Decode	368
5-34	PMU Register Summary	376
5-35	PMU Control Bus data Map	378
5-36	CHAP Command N Register Bit Definition	388
5-37	CHAP Events N Register Bit Definition	391
5-38	CHAP Status N Register Bit Definition	393
5-39	CHAP Data N Register Bit Definition	395
5-40	SlowPort Register Map	395
5-41	Corresponding Clock Division Values with Respect to the Register Values (for IXP2xxx rev A)	396
5-42	Corresponding Clock Division Values with Respect to the Register Values (for IXP2400 rev B)	397
5-43	Corresponding Clock Division Values with Respect to the Register Values (for IXP2800 rev B)	397
5-44	MSF Register Summary	408
5-45	Number of Elements per RBUF or TBUF Partition	417
5-46	New Port Status to be saved based on currently saved value and new value received on TSTAT	421
5-47	List of RX_DESKEW_# Registers	433
5-48	RBUF High Water Marks	439
5-49	Rx_Thread_Freelist Use	440
5-50	CSIX TBUF_ELEMENT_CONTROL_A_#	447
5-51	CSIX TBUF_ELEMENT_CONTROL_B_#	448
5-52	SPI-4 TBUF_ELEMENT_CONTROL_A_#	448
5-53	SPI-4 TBUF_ELEMENT_CONTROL_B_#	449
5-54	List of RX_PHASEMON_# Registers	455
5-55	IXP2400 MSF Address Map	462
5-56	IXP2400 MSF Allowable Major Bus Modes	469
5-57	IXP2400 Rx Mode Programming	491
5-58	UTOPIA Transmit Control Word Format	499
5-59	POS-PHY Transmit Control Word Format	500
5-60	CSIX Transmit Control Word Format	501
5-61	PCI Configuration Register Map	502
5-62	PCI MEM Space CSR Register Map	513
5-63	Descriptor Format:	520
5-64	Operation of Unlinked Descriptor	521
5-65	How Window Sizes are Determined (PCI_SRAM_BAR)	525
5-66	How Window Sizes are Determined (PCI_DRAM_BAR)	526
5-67	Intel XScale™ Core Gasket Configuration Register Map	541
5-68	Hash Operation/Result Register Map	556



5-69	Break Point Register Map for the Intel XScale™ Core	561
A-1	UCA Warnings.....	567

Introduction

1

1.1 About this Document

This manual serves as a reference for microcode programming the Intel® IXP2400 and Intel® IXP2800 Network Processors. The intended audience for this book is Developers and Systems Programmers.

The book is organized as follows:

[Section 2, “Assembler”](#) describes the assembler.

[Section 3, “MEv2 Instruction Set”](#), describes the microinstruction set and provides example microcode.

[Section 4, “Address Maps”](#), provides the address maps for the MEs, PCI and the Intel XScale® core.

[Section 5, “Control and Status Registers \(CSRs\)”](#), describes the internal registers and provides examples of their use.

[Appendix A, “UCA Warnings”](#), lists the UCA Warnings and error messages.

Note: For a detailed technical description of the IXP2800 Network Processor, refer to the *IXP2800 Network Processor Hardware Reference Manual*. Similarly, refer to the *IXP2400 Network Processor Hardware Reference Manual* for a detailed technical description of the IXP2400 Network Processor.

1.2 Related Documentation

Further information is available in the following documents:

IXP2800 Network Processor Datasheet - Contains summary information on the IXP2800 including a functional description, signal descriptions, electrical specifications, and mechanical specifications.

IXP2400 Network Processor Datasheet - Contains summary information on the IXP2400 including a functional description, signal descriptions, electrical specifications, and mechanical specifications.

IXP2400/IXP2800 Network Processor Development Tools User’s Guide - Describes the Workbench and the development tools you can access through the use of the Workbench.

IXP2800 Network Processor Hardware Reference Manual - Contains detailed hardware technical information of the IXP2800 Network Processor for designers.

IXP2400 Network Processor Hardware Reference Manual - Contains detailed hardware technical information of the IXP2400 Network Processor for designers.

Assembler

2

This chapter describes the microcode assembler.

2.1 Acronyms

Table 2-1 lists common acronyms used in this chapter.

Table 2-1. Acronym Definitions

Acronym	Description
CAP	CSR Access Proxy: IXP2800/IXP2400 functional unit containing majority of CSRs
GPR	General Purpose Register
MSF	Media Switch Fabric
UCA	Microcode Assembler
UCLD	Microcode linker that links together the .list files from multiple microengines
microword	Microcode instruction word
MEv1	Microengine Version 1 (e.g. IXP1200)
MEv2	Microengine Version 2 (e.g. IXP2800 and IXP2400)

2.2 Definitions

For the purposes of this document, the word *scope* of a virtual register refers to that portion of the flattened input source in which it is possible to refer to that virtual register. The *live-range* (or *lifetime*) of a virtual register refers to that portion of the code where that register contains a value that will be used later. Generally, the live-range extends from when the register is set to where that value is last used.

Note that the scope and live-range may not coincide. In particular, the live-range may extend outside of the scope due to, for example, a subroutine call. Even if the subroutine is outside of the scope of the register, the register may be live within the subroutine. Similarly, the live-range may be smaller than the scope.

Transfer registers come in two variants. One is a *read* or *in* transfer register. These are registers that may be used as the source of an ALU operation. The name comes from the fact that they typically are used for “read” I/O operations. The other is a *write* or *out* transfer register. These may be used as a destination of an ALU operation. They are typically used for “write” I/O operations. A virtual register may be defined as a *read* transfer register, as a *write* transfer register, or as a *R/W* (or *both*) register. In the first two cases, the virtual register is allocated from either the read or write banks of transfer registers. In this third case, the virtual register is allocated in both the read and write banks at the same address. These are needed for I/O operations that do both reads and writes at the same time. Historically, all transfer registers fell into this category.

2.3 Source File Elements

A source file (.uc) must be created before the assembly process can begin. The .uc file contains three types of elements:

- Instructions: Consists of an opcode and arguments and generate a microword in the .list file.
- Directives: Pass information either to the preprocessor, assembler, or to downstream components (e.g., the linker) and generally do not generate microwords.
- Comments: Ignored in the assembly process.

The elements in the source are case insensitive.

A *microword* is the result of assembling one instruction.

2.3.1 Instructions

Instruction lines in the source code generate microwords in the output. They can be preceded by zero or more labels and can be followed by an optional set of parameters followed by optional modifiers. Instruction lines can span multiple physical lines of input.

A label is a symbol representing an instruction address that is resolved by the assembler (e.g., start#). A label is composed of a string of alphanumeric characters (including “_”) which ends in the pound sign (#) followed by a colon (:). A reference to a label (e.g., in a branch instruction) would omit the colon character because it is referencing a label defined elsewhere.

By convention, labels start in column 1, though this is not a requirement for the assembler. You can use none or as many labels as you want for each instruction. The only restriction is that each label must have a unique name.

All labels for a given instruction must be defined before the actual instruction specification.

2.3.2 Directives

Directives pass information to the assembler or linker, but they generally do not generate microwords in the output. Directives start with the directive name optionally followed by parameters. Directives cannot span multiple lines.

2.3.3 Comments

There are two forms of comment. One comment form starts with a semicolon (;) character and runs to the end of the line. Thus, each new comment line must start with a semicolon. The semicolon may start anywhere on any line. Comments on a line that also contains all or part of an instruction are associated with that instruction. Lines containing comments alone are associated with the next instruction following it. Comments beginning with a semicolon appear in the output file.

C-style comments (e.g., //comment or /* comment */) are also supported, but these comments are removed and do not appear in any output file. Comments attached to token expansion definitions (e.g., #define) and comments associated with macro parameters do not appear in the output.

2.4 Block Structure

The source code is logically broken into a hierarchy of blocks. That is, blocks may contain sub-blocks, but a block cannot be partially contained in a higher-level block.

Blocks are explicitly delimited by “.begin” and “.end” directives,

```
.begin  
.end
```

Or in the older supported directives,

```
.local  
.endlocal
```

Or by directives that define a subroutine.

```
.subroutine  
.endsub
```

2.5 Assembly Process Steps

As shown in [Figure 2-1](#), invoking the assembler results in a two-step process composed of preprocessing and assembly steps. The preprocessor step takes a .uc file and creates a .ucp file for the assembler. The assembler takes a .ucp file and creates an intermediate file with the file name extension of .uci. The .uci file is used by the assembler to create the .list file and provides error information that may be used in resolving semantic problems (such as register conflicts) in the input file.

The assembler performs the following functions in converting the .uc file to a .list file:

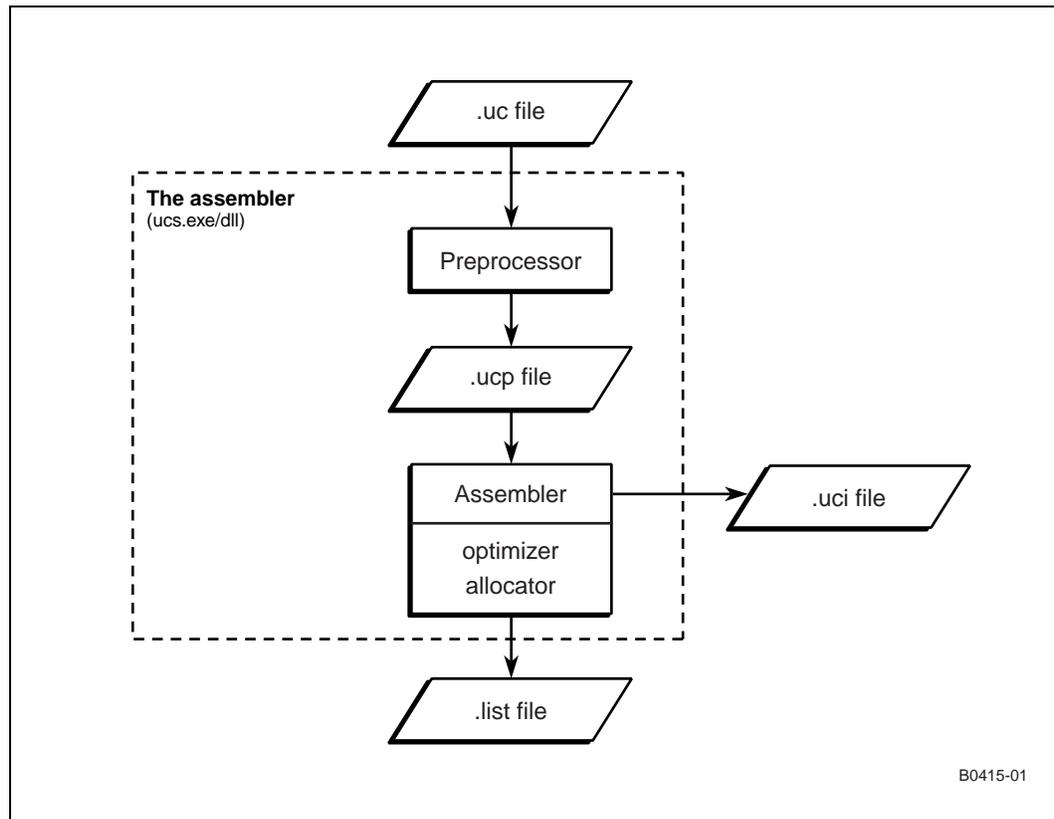
- Checks instruction restrictions.
- Resolves symbolic register names to physical locations.
- Optimizes the code, by inserting defer[] optional tokens.
- Resolves label addresses.
- Translates symbolic opcodes into bit patterns.

The preprocessor is invoked from within the assembler. Command line options are available when invoking UCA.exe (or UCA.dll via the workbench).

```
uca [options] microcode_file microcode_file...
```

For detailed information on Assembler Command Line Options, refer to the *IXP2400/IXP2800 Network Processor Development Tools User's Guide*.

Figure 2-1. Assembly Process



2.6 Assembler Preprocessor

The preprocessor is invoked automatically by the assembler to transform a program before actual assembly. The preprocessor provides six separate facilities that you can use as you see fit:

- Inclusion of files. These are files of declarations that can be substituted into your program.
- Macro expansion. You can define macros, which are abbreviations for arbitrary fragments of assembly code, and then the preprocessor will replace instances of the macros with their definitions throughout the program.
- Conditional compilation. Using special preprocessing directives, you can include or exclude parts of the program according to various conditions.
- Line control. If you use a program to combine or rearrange source files into an intermediate file which is then assembled, you can use line control to inform the assembler of where each source line originally came from.
- Structured Assembly. You can organize the control flow of the ME instructions into structured blocks as opposed to a sea of goto statements.

- Token Replacement. You can use causes instances of an identifier to be replaced with a token string.

2.6.1 Preprocessor Reserved Labels

The preprocessor generates labels during macro expansion and during conditional assembly. Avoid using labels with these prefixes to avoid confusion with those generated by the preprocessor and to avoid the possibility of multiple label definitions. In the following table, nnn represents a three-digit decimal number.

Mnnn_ Prefix used for macro references not preceded by a label.

Innn_ Prefix used for labels for structured assembly constructs.

2.6.2 Preprocessor Operation

The preprocessor is a simple macro processor that processes the source file before the it is assembled. It is important to have a basic understanding of how the preprocessor operates to understand how directives interact with one another. This section provides a brief overview.

During the initial reading of an input file, there are three occasions when the file is read but not processed:

- Within a #if...#endif clause, if the text is being skipped.
- Within a macro definition.
- Within the body of an assembly loop (e.g., #repeat).

In each of these cases, no processing of directives takes place, with the exception of the directive that ends that context. Constructs of a similar type may nest, however, so that if within a macro definition there is another macro definition, the first macro definition will not end until the second (i.e., the matching) .endm is reached. Within a particular context, other directives are ignored. For example, if a macro definition had a #if without a matching #endif, an error would not be reported until the macro was referenced (expanded). So these constructs can be nested within each other, but they cannot be only partially contained within each other. It would be an error, for example, to put an unmatched #if within one macro and the “matching” #endif in another.

Lines that are being processed have expandable tokens expanded. Then macro references are expanded. This means that an expandable token used as an argument in a macro reference is expanded at the time of the reference, not when it is used within the body of the macro.

Table 2-2. Summary of Preprocessor Directives (Sheet 1 of 2)

Directive	Arguments Expanded?	Description
#include	No	Start reading lines from another file.
#define	No	Define an expandable token.
#define_eval	Yes	Define an expandable token as the result of evaluating a constant expression.
#undef	No	Undefine an expandable token.
#ifdef, #ifndef	No	Conditionally skip following lines.

Table 2-2. Summary of Preprocessor Directives (Sheet 2 of 2)

Directive	Arguments Expanded?	Description
#if, #elif	Yes, including defined(name)"	Conditionally skip following lines based on a constant expression.
#else, #endif	N/A	Conditionally skip following lines.
#macro	No	Start defining a macro.
#endm	N/A	Finish defining a macro.
#repeat, #while	Yes	Repeat following lines based on a constant expression.
#for	No	Repeat following lines based on a constant expression.
#endloop	N/A	End repeated lines.
.if, .elif	Yes	Generate branch instructions.
.if_unsigned, .elif_unsigned	Yes	Generate branch instructions
.else, .endif	N/A	Generate branch instructions.
.while	Yes	Generate branch instructions.
.while_unsigned	Yes	Generate branch instructions.
.endw	N/A	Generate branch instructions.
.repeat	N/A	Generate branch instructions.
.until	Yes	Generate branch instructions.
.until_unsigned	Yes	Generate branch instructions.
.break, .continue	N/A	Generate branch instructions.

2.6.3 Constant Expressions (const-expr)

Constant expressions are expressions that evaluate to a constant. Generally, after the assembler performs token substitution, the expression consists only of numeric constants and operators. The exception to this is a number of preprocessor functions that take identifiers as arguments and that return identifiers or numeric constants as values. Within an instruction, wherever a constant is valid, you can use a constant expression that is surrounded by parenthesis. The parentheses are needed to differentiate expressions from tokens such as "B-A" which should not be evaluated. For some directives, the parenthesis may be omitted, but it is generally a good idea to use them. Wherever the term **const_expr** appears in this manual, it can be replaced with **(const_expr)**, where const_expr is one of the following:

- (const)
- (const-expr bin-op const-expr)
- (unary-op const-expr)
- (const_expr ? const_expr : const_expr)
- function (token, token, ...)

2.6.3.1 Preprocessor Binary & Unary Operators

The following binary and unary operators are supported within constant expressions. Operator precedence is the same as defined for the C programming language.

Table 2-3. Binary & Unary Operators

Type	Operator	Associativity	Comment
unary-ops	! ~ + - (unary)	Right to left	
bin-ops	* / %	Left to right	
	+ -	Left to right	
	<< >>	Left to right	
	< <= > >	Left to right	These relational operators assume signed 32-bit values
	== !=	Left to right	
	&	Left to right	
	^	Left to right	
		Left to right	
	&&	Left to right	
		Left to right	
	?:	Right to left	
,	Left to right		

2.6.3.2 Preprocessor: Functions

The following functions are supported within constant expressions. These functions, with the exception of "defined", operate on the results of expanding tokens and evaluating expressions.

Note that in expanding .if and .elif, the defined(name) construct is replaced by a 0 or 1 as appropriate.

Table 2-4. Functions (Sheet 1 of 2)

Function	Description
IS_IXPTYPE(type)	Returns non-zero if the targeted processor type is only of the type given by the parameter type and no other. The calculation performed is "!(__IXPTYPE & ~(type))". Note: the parameter type can be an expression such as "(type1 type2)". This function should be used in conjunction with the predefined symbols described in Section 2.6.8 .
isnum (token)	Returns 1 if the token expands to a numeric constant, otherwise it returns 0.
isimport(token)	Returns 1 if the token begins with "i\$", which indicates that it is an import variable. Otherwise it returns 0. Note: the .import_var directive will generate a warning if an import variable does not begin with "i\$" and it was used in an isimport() call.

Table 2-4. Functions (Sheet 2 of 2)

Function	Description
streq (token1, token2)	Returns 1 if both tokens are identifiers which match, or if both are numeric constants which match; otherwise it returns 0.
strstr (token1, token2)	Returns the index of the first occurrence of token2 in token1 (starting with 1). If token2 is not found in token1, then it returns a value of 0. If either token is not an identifier, it returns a value of -1.
strlen (token)	Returns the number of characters in token. If the token is not an identifier, it returns -1.
strleft (token1, token2)	Returns an identifier consisting of the leftmost token2 characters of token1. If token1 is not an identifier or token2 is not numeric, the identifier "error" is returned.
strright (token1, token2)	Returns an identifier consisting of the rightmost token2 characters of token1. If token1 is not an identifier or token2 is not numeric, the identifier "error" is returned. Note that strright(token, -len) is essentially equivalent to strright(token,strlen(token)-len).
defined(token)	Evaluates to 1 if the token is a symbol defined within the preprocessor or 0 otherwise.
log2(arg, round) log2(arg)	Returns the log-based-2 of arg as an integer. The round argument is optional and if omitted, it defaults to 0. Refer to Section 2.6.3.4, "LOG2() Function" for a detailed description of this function.
mask(sig)	Evaluates to 1 if sig is a single signal and 3 if sig is a double signal. For a detailed explanation of this function, refer to Section 2.8.8.1, "Accumulating Results for ctx_arb[-]" .

The constant expression function “strright(token,len)” originally meant to take the *len* characters from the right-most position of *token*. Now, if *len* is ≤ 0 , it will mean to drop the left-most $-len$ characters. For example:

```
strright(abcdef, 2) ⇒ ef ; original behavior
strright(abcdef, -2) ⇒ cdef ; new behavior
```

2.6.3.3 STRING Operator

One of the limitations of the “string functions” within the preprocessor constant-expression parsing is that they operate on identifiers, not true strings. This has practical implications. For example, a macro may want an ALU operation passed in, and it may want to do something different based on whether that operation allows a shift or not. The problem is that the operation cannot be compared with streq, because the name of some of the operations is not a valid identifier. To address this, there is a new operator defined by single quotes.

This operator will return a valid “identifier”, composed of the text enclosed by the quotes after token expansion, that is, after macro arguments are expanded. However, the identifier is created before any of the “arguments” are evaluated based on normal expression rules.

More precisely, the function will return an “identifier” formed by taking all of the (expanded) text between the single quotes, minus any leading and trailing white space.

For example, one could write:

```
#macro test(arg)
  #if (streq('arg', 'b-a'))
  ...
```

Note that the string operator was used both on the arg and in the comparison string. This points out that the argument to the string operator may be a constant rather than an expandable token. One detail to note: the leading and trailing white space is deleted, but interior white space is not.

In the context of constant-expressions, an identifier can also be created using double-quotes. This behaves the same as the single-quoted version defined above, except that leading and trailing white space is not removed. This would probably be used typically to construct an identifier consisting only of white space, e.g. `strstr('token', " ")`.

Note that text within double quotes is not token-expanded.

Here are some examples:

```
#define A 1
#define B 2
' A + B ' ; evaluates to "1 + 2"
' 1 2 3 ' ; evaluates to "1 2 3"
" A + B " ; evaluates to " A + B "
" 1 2 3 " ; evaluates to " 1 2 3 "
```

In this case, the string function would evaluate to "1 + 2", not "3". This illustrates the rule that arguments are expanded but expressions are not evaluated.

2.6.3.4 LOG2() Function

The function `log2()` can be used within constant expressions. It takes two arguments, the second of which is optional:

```
log2 (arg, round)      or      log2 (arg)
```

arg: Numeric value (taken as an unsigned value) whose log-2 value is desired
round: Optional numeric value determining how arg is to be rounded

The function returns the log-based-2 of arg as an integer. If the round argument is not supplied, then the default rounding is 0. If arg is a power of two, then the same value is returned regardless of round. If arg is not a power of two, the behavior depends on round, which is described in the following tables:

Condition	Results when arg is not a power of two
round < 0	Round result down to next smaller integer.
round = 0	Generate an error.
round > 0	Round result up to next larger integer.

Condition	Results when arg is zero
round < 0	-1
round = 0	Generate an error.
round > 0	0

Here are some examples.

Table 2-5. Examples of log2() Function

arg	log2(arg) == log2(arg,0)	log2(arg, -1)	log2(arg,1)
0	error	-1	0
8	3	3	3
10	error	3	4
0xFFFFFFFF == -1	error	31	32

2.6.3.5 Preprocessor Function Examples

The following examples show the usage of the functions.

The defined(token) constant expression evaluates to 1 if the token is a symbol defined within the preprocessor, or 0 otherwise. Typical usage would be:

Examples: Defined(token)

```
#if (defined(FOO) || defined(BAR))
```

Examples: ISNUM

```
#macro assign[reg, val]
  #if (isnum(val))
    // value is a numeric constant
    immed[reg, val]
  #else
    // assume arg is the name of a register
    alu[reg, --, b, val]
  #endif
  ...

```

Examples: STREQ

```
#macro something[type]
#if (streq(type, sync))
...
/* This allows the application to specify type as the string
 * "sync", assuming that the user has not #defined sync to be
 * something else. So the application could call this macro as:
 * something[sync]
 * or
 * something[async]
 */
```

Examples: STRSTR

```
#macro somethingelse[reg]
#if (strstr(reg, @) > 0)
/* reg is absolute */
#else
/* reg is relative */
#endif
```

2.6.4 Macros and Expansion Token Restriction

Macros and expansion tokens share the same name space; therefore, it is invalid to have a macro with the same name as a #define token.

2.6.5 Syntax for Argument and Token lists

In the preprocessor, several places exist where commas are used in separating items in a list. For example:

```
#for identifier [arg1, arg2, ...]
```

and macro references:

```
macro[arg1, arg2, ...]
```

In both cases, commas can be included in the items list as long as they are enclosed in a matching set of parentheses or brackets. For example, the directive:

```
#for id[item1, foo(bar,bif), immed[reg,32]]
```

would be expanded with id taking three values:

```
item1  
foo(bar,bif)  
immed[reg,32]
```

The assembler does not interpret the “,” and take “bif)” or “immed[reg” as values.

2.6.6 Leading and Trailing Spaces in Macros

Leading and trailing spaces in macro arguments are automatically removed by the assembler. For example:

```
marco[ arg1, arg2 ]
```

would be translated by the assembler as

```
macro[arg1,arg2]
```

2.6.7 Environment Variables

The following environment variables are recognized by the assembler:

UCA_INCLUDE: A list of directories to be added to the include path. The list is separated by semicolons:

```
dir1;dir2;dir3...
```

and is appended after the directories supplied on the command line.

2.6.8 Predefined Processor Type and Revision Symbols

The preprocessor defines the symbol `__IXPTYPE`, which identifies the target processor type(s) for the code being assembled. It is defined to be some combination of the various processor type symbols also defined by the preprocessor. Table 2-6 lists the processor type symbols and their meanings.

The preprocessor also defines two symbols `__REVISION_MIN` and `__REVISION_MAX`, which specify the target processor revision for the code being assembled. The values of these symbols are defined according to the command line values or by the Workbench assembler settings. The revision values consist of an 8-bit value. The upper 4-bits correspond to the major revision number (which is a letter, such as “A”, “B”, etc) while the lower 4-bits correspond to the minor revision number (such as 1, 2, 3, etc.).

Table 2-6. Processor Type Symbols

Symbol	Meaning	Type
__IXPTYPE	Value is determined by command line arguments. It takes on some combination of the following values.	variable
__IXP2400	Processor Type IXP2400	1-bit set
__IXP2800	Processor Type IXP2800	1-bit set
__IXP28XX	IXP2800	n-bits set
__IXP2XXX	All MEv2 network processors.	n-bits set

All these symbols, other than the first one, are pure constants that have only one bit set (for the processor-specific ones). The __IXPTYPE symbol indicates the processor types for which code is being generated. Its value can consist of a single processor type or a combination of types.

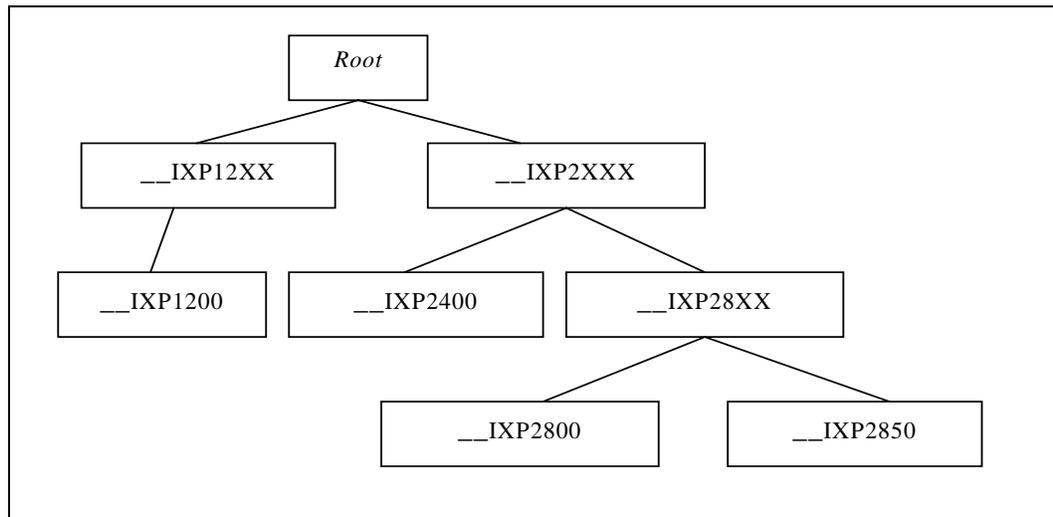
The preprocessor also defines symbols for the various revision values. The revision symbols are listed in [Table 2-7](#).

Table 2-7. Revision Symbols

Symbol	Meaning	Value
__REVISION_MIN	Minimum processor revision for code being assembled.	Variable. Default is 0x00
__REVISION_MAX	Maximum processor revision for code being assembled.	Variable. Default is 0xff
__REVISION_A0	A0 revision.	0x00
__REVISION_A1	A1 revision.	0x01
__REVISION_B0	B0 revision.	0x10
__REVISION_B1	B1 revision.	0x11
etc...		

Constant values can be thought of as a tree, as shown in [Figure 2-2](#)

Figure 2-2. Processor Type Constant Values



The leaf nodes represent values with a single bit set while non-leaf nodes represent the union (bitwise-OR) of the nodes below them.

A default value can also be set using the environment variable UCA_IXPTYPE. For example:

```
set UCA_IXPTYPE=ixp2400
```

These predefined symbols should be used with the IS_IXPTYPE(type_expression) constant expression function described in [Section 2.6.3.2](#).

Examples: Different Code Sequence for IXP2400 or IXP2800

```

#if (IS_IXPTYPE(__IXP28XX))
    // code for IXP2800
#elif (IS_IXPTYPE(__IXP2400))
    // code for IXP2400
#else
    #error
#endif
  
```

Examples: Code Only Works on MEv2

```
#if (!IS__IXPTYPE(__IXP2XXX))
    #error
#endif
```

Any code targeted at a specific revision of the IXP2400/IXP2800 Network Processor should make use of the predefined `__REVISION_MIN` and `__REVISION_MAX` symbols. For example, if the code is designed to exploit certain features found only in processor revisions A1 or higher, then the `REVISION_MIN` symbol can be used along with the `#error` directive to abort the assembly process.

For example,

Revision Number	Hex Value
-----	-----
A0:	0x00
A1:	0x01
A2:	0x02
B0:	0x10
B1:	0x11
B2:	0x12
etc.	

The default values for `__REVISION_MIN` and `__REVISION_MAX` are 0x00 and 0xff, respectively.

In addition to the `__REVISION_MIN` and `__REVISION_MAX` symbols, the assembler also predefines symbols of the form:

```
#define __REVISION_A0 (0x00)
#define __REVISION_A1 (0x01)
...
#define __REVISION_B0 (0x10)
#define __REVISION_B1 (0x11)
...
etc.
```

Examples: Revision Symbol Use

```
#if (__REVISION_MIN < __REVISION_A1)
#error "This feature is not supported on revision(s) prior to A1"
#else
; version specific code here
#endif
```

For more information on setting the processor type or revision, please refer to the *IXP2400/IXP2800 Network Processor Development Tools User's Guide*.

2.6.9 Predefined Import Variables

The Predefined Import variables are detailed in [Table 2-8](#). For more information on import variables, refer to [Section 2.11.8](#), “Import Variable (.import_var)”.

Table 2-8. Predefined Import Variables

Name	Definition
__CHIP_ID and i\$__CHIP_ID	This corresponds to the IXPTYPE valued defined in Section 2-6 , “Processor Type Symbols” and is derived from the PRODUCT_ID register.
__CHIP_REVISION and i\$__CHIP_REVISION	This gives the chip revision as defined in Section 2.6.8 , “Predefined Processor Type and Revision Symbols”, which is derived from the “PRODUCT_ID” register.
__UENGINE_ID and i\$__UENGINE_ID	This varies from 0x00 - 0x07 and 0x10 - 0x17 for the IXP28xx and from 0x00 - 0x03, 0x10 - 0x13 for the IXP2400.

2.7 Preprocessor Usage Techniques

This section contains techniques that you may find useful in writing ME software.

2.7.1 Branching into a Macro

There are occasions where the code might branch into a macro. This generally should be avoided as the macro then becomes more difficult to modify, but it might be necessary.

Examples: An Incorrect Method

An incorrect method would be to pass the label in, as follows:

```
#macro bad1 [lab, ...]
    ...
    lab alu[...]
    ...
#endm
```

and then to use it as:

```
bad1 [mylab#, ...]
    ...
    br [mylab#:]
```

Which doesn't work. What happens is that the macro expands its arguments, so that it gets (in the example above):

```
mylab#: alu[...]
```

The preprocessor then realizes that a label is being defined within a macro expansion and augments the name to make it unique. What you then get would be:

```
M001_mylab#: alu[...]
```

Examples: A Correct Method

The correct way to handle this circumstance is to define the label as normal within the macro, e.g.,:

```
#macro good1[...]  
...  
lab#: alu[...]  
...  
#endm
```

When the macro is referenced, prefix the macro reference with a label. This provides a way to identify which macro is being referenced. In this example, the macro reference would appear as:

```
use1#: good1[...]
```

To branch into the macro, one takes advantage of the fact that if the macro reference is preceded by a label, that label (followed by an underscore) is used as the label prefix of the augmented label. Thus, one would branch into the macro as:

```
br[use1_lab#]
```

where use1 is the label of the reference (minus the #) and lab is the label within the macro definition. This technique may be extended to jumping into nested macros by preceding the macro reference within the macro definition with a label.

2.7.2 Constructing Names from Numbers

There may be times where a repetitive task is to be done on a series of registers, where the register name is formed by a base name and a number. For example, in the case where one wanted to generate the lines:

Examples: Example Case

```
alu[reg1, 0, +, reg, <<0]  
alu[reg2, 1, +, reg, <<2]  
alu[reg3, 2, +, reg, <<4]  
alu[reg4, 3, +, reg, <<6]  
alu[reg5, 4, +, reg, <<8]  
alu[reg6, 5, +, reg, <<10]
```

One way to do this would be:

Examples: Example case - Alternate Method 1

```
#macro oneway[reg, const, shift]
alu[reg, const, +, reg, shift]
#endm
oneway[reg1, 0, <<0]
oneway[reg2, 1, <<2]
...
```

Another way would be to take advantage of the relationship between the numeric portion of the name, the constant, and the shift value. You could write:

Examples: Example case - Alternate Method 2

```
#macro anotherway[num]
#define_eval const num-1
#define_eval shift const*2
alu[reg/**/num, const, +, reg, <<shift]
#endm
anotherway[1]
anotherway[2]
anotherway[3]
...

<or>

#for cnt [1, 2, 3, 4, 5]
anotherway[cnt]
#endloop
```

The problem with the expression `reg/**/num` is that you want to attach the value of `num` to alphabetic characters. It is easy to do this kind of attachment to non alphabetic characters, (e.g., `<<shift`), but if we were to write `regname_num`, it would be taken as a single token that would not be expanded. If we were to write `reg num`, for example “`anotherway[3]`”, it would expand to `reg 3`, which would also be incorrect.

To get the correct value, we take advantage of the fact that the preprocessor removes C-style comments. In this example, the comment is a minimal C-style comment. It serves to delimit the `reg` and the `num`, but since it is removed, there is nothing left between the expanded texts.

2.8 Registers and Signals

The assembler resolves symbolic register names into physical register addresses. The following sections describe the details of registers and signals.

2.8.1 Register Naming Conventions

Registers are specified symbolically using a string of alphanumeric characters (including “_”). The first character of a register name cannot be numeric. The bank to be accessed (such as GPR, SRAM XFER, or DRAM XFER) and the addressing mode (context-relative or absolute) is determined by prefixing the register name with the reserved characters “@” and “\$”. Absolute addressing is only supported for GPR registers; it is not supported for SRAM and DRAM transfer registers.

Register types are defined by prefixes applied to the register names. Registers have a type based on the name. The table below shows the prefix (in bold) that is applied to register names to specify the type. The word “reg” is the user specified name of the register. Local memory is shared by all contexts using an index register and does not support relative or absolute names. Transfer and neighbor registers can also be accessed globally using index registers.

Register Type	Relative Name	Absolute Name
GPR	reg	@reg
SRAM Transfer	\$ reg	not available
DRAM Transfer	\$\$ reg	not available
Next Neighbor	n \$reg ^a	not available

a. Named next neighbor registers are not supported in restricted addressing mode (refer to [Section 2.8.1.1, “Indexed Registers”](#) and [Section 3.1.1, “Restricted and Unrestricted Src and Dest Operands”](#)).

One reason for this paradigm is to allow macros to determine the type of register being passed in as an argument.

Note that whether an SRAM or DRAM transfer register is allocated strictly out of the read/in or write/out registers is not considered “type” information and is not indicated by the name.

2.8.1.1 Indexed Registers

The MEv2 allows access to some of the register types by using an index register. An index register is a local CSR that points to an address in the related register file. The actual register that is addressed can be accessed in a manner similar to that of “normal” registers.

Generally, the index is set using the `local_csr_wr` instruction¹. The exception is the index register used for writes to the neighbor register. In this case, the CSR is located in the neighbor ME and is not visible to the writer (although the neighbor can access it). Typically, it is initialized by the neighbor and then the neighbor registers function as a FIFO, with the read/write index registers never being directly set again (they are always advanced indirectly via post-increment).

Local memory can only be accessed through indexed registers; it does not support the use of “named” registers. In the case of local memory, there are two independent index registers (zero and one). The contexts in an ME can either share the same two registers, or each context can reference its own set. For all other indices, each context within an ME shares the same index.

1. See [Section 5, “Control and Status Registers \(CSRs\)”](#) for details on which local CSRs are used for which indices.

Neighbor registers can be accessed by name or by an index, but the programmer will not typically access the registers using both names and index in the same program¹. Transfer registers can also be accessed by name or by an index; however, it is more likely that you will access the registers using both names and index in the same program.

Each of the indices supports a number of additional features, including:

- Post-increment
- Post-decrement
- Offsetting

The feature set of the different indices are shown in the following table:

Register Type	Register Name	Post-increment	Post-decrement	Offsetting	Local CSR Name for Index Register ^a
Local Memory	*I\$index0 *I\$index1	*I\$index0++ *I\$index1++ ^b	*I\$index0-- *I\$index1-- ^b	*I\$index0[n] *I\$index1[n] ^c	ACTIVE_LM_ADDR_0 ACTIVE_LM_ADDR_1
Next Neighbor Fifo	*n\$index	*n\$index++ ^d	N/A	N/A	NN_PUT NN_GET
SRAM Transfer	*\$index	*\$index++	*\$index--	N/A	T_INDEXI
DRAM Transfer	*\$\$index	*\$\$index++	*\$\$index--	N/A	

- a. Refer to Section 5 of this manual for a complete list of Local CSR Names.
- b. Post increment and Post decrement are supported for unrestricted addressing only. See Section 3.1.1, "Restricted and Unrestricted Src and Dest Operands"
- c. For offsetting, n = 0 to 15 for unrestricted addressing. See Section 3.1.1, "Restricted and Unrestricted Src and Dest Operands"
- d. Optional when used as a source and required when used as a destination.

Caution: The SRAM and DRAM indexing operations use the same local CSR as the index to these memories.

The use of an indexed register reference is indicated by the leading asterisk ("*") in the register name. After that comes the normal type prefix and then the keyword "index". In the case of local memory, the keyword "index" is followed by a "0" or "1" to identify which Local Memory CSR index register to use. Post-increment or post-decrement is indicated by appending "++" or "--" to the register name. For example, "*n\$index++".

Offsetting refers to addressing a word (4-bytes) at a fixed offset from the word addressed by the Index local CSR. The offset is a constant in the range from 0 to 15 words. When an offset is used, the offset is bit-wise ORed into the address. This means that the address register must be aligned on an appropriate boundary for offsetting to work². Offsetting is indicated by appending a numeric constant surrounded by square brackets to the register name. An example would be "*I\$index0 [3]".

Offsetting cannot be used with post-increment or post-decrement.

1. The index implements a FIFO that will eventually overwrite any named register.
2. Alignment needs to be maintained by the programmer. The assembler cannot check for proper alignment.

An index register is the only way that the local memory can be accessed. Local memory does not support the use of "named" registers as do the other register files. In the case of local memory, there are two independent index registers for each context and one set for the active context (the context that is currently executing).

In the case of local memory, the two index registers are referenced within the instructions using the keywords *I\$index0 and *I\$index1. These keywords always refer to the active set. The ME can be put into a mode where all contexts share the same two registers (the active set), or each context uses its own set. This is specified using the assembler directives `local_mem0_mode`, and `local_mem1_mode` (refer to [Section 2](#) for more information on these directives). When an ME is set up to have each context use their own two index registers and a context begins executing, its set of registers are loaded into the active set and when the context goes to sleep, the active set are saved back to context's set. In the mode where all contexts share the same two registers, the active set is only set that is used.

The local memory index registers are loaded using the `local_csr_wr` instruction. The following register names can be used to access the two independent index registers for the active context.

```
ACTIVE_LM_ADDR_0           ACTIVE_LM_ADDR_1
ACTIVE_LM_ADDR_0_BYTE_INDEX ACTIVE_LM_ADDR_1_BYTE_INDEX
```

For the contexts that are not active, users can access their local registers by first setting `CSR_CTX_POINTER` to the correct context number. Then, the following register names can be used to access the two independent index registers of the specified context.

```
INDIRECT_LM_ADDR_0           INDIRECT_LM_ADDR_1
INDIRECT_LM_ADDR_1_BYTE_INDEX INDIRECT_LM_ADDR_1_BYTE_INDEX
```

Refer to [Section 5](#) for more information on all these registers.

The Next Neighbor registers can use the index register in support of Next Neighbor Rings. When the destination register is specified as *n\$index++ the `NN_PUT` index register is used to perform a put operation. When the source register is specified as *n\$index++ the `NN_GET` index register is used to perform a get operation.

The entire transfer register set can be accessed by a context using the Transfer Register Index Registers (`T_INDEX`). A register number (as shown in [Table 2-9](#)) is written to the `T_INDEX` register using the `local_csr_wr` instruction. Then in transfer register is read or written using the notation shown in the [Table 3-4](#) to specify either a source (for a write) or a destination (for a read).

Table 2-9. Registers Used By Contexts in Context-Relative Addressing Mode

Number of Active Contexts	Active Context Number	GPR Absolute Register Numbers		S Transfer or Neighbor Index Number	D Transfer Index Number
		A Port	B Port		
8 (Instruction always specifies Registers in range 0-15)	0	0-15	0-15	0-15	0-15
	1	16-31	16-31	16-31	16-31
	2	32-47	32-47	32-47	32-47
	3	48-63	48-63	48-63	48-63
	4	64-79	64-79	64-79	64-79
	5	80-95	80-95	80-95	80-95
	6	96-111	96-111	96-111	96-111
	7	112-127	112-127	112-127	112-127
4 (Instruction always specifies Registers in range 0-31)	0	0-31	0-31	0-31	0-31
	2	32-63	32-63	32-63	32-63
	4	64-95	64-95	64-95	64-95
	6	96-127	96-127	96-127	96-127

Two registers of the same type other than GPR cannot appear as source operands in a single instruction, but two registers of the same type can appear with one being a source and one being a destination. This raises the question of what happens if in this case one wishes to apply an increment/decrement operator to that register. The rule that the assembler uses is that when the same index register is used as both a source and destination, any increment/decrement operator must be applied to the destination usage. Usage on the source or on both will result in an error¹.

Thus:

```
alu[*1$index0++, 1, +, *1$index0 ]
```

would be valid, but

```
alu[*1$index0 , 1, +, *1$index0++]
alu[*1$index0++, 1, +, *1$index0++]
```

would not. This is to prevent confusion in people who are looking at the code and who might think that the first bad case is writing to the next register after the one being read, and who might think that the second bad case is incrementing the index register twice.

Note that neighbor registers have different read and write pointers, so both of the following are valid:

```
alu[*n$index++, 1, +, *n$index]
alu[*n$index++, 1, +, *n$index++]
```

It is allowed that the same local memory index can be offset differently as source and destination in the same instruction. Thus, the following is valid:

```
alu[*1$index0[3], 1, +, *1$index0[4] ]
```

1. Conceptually, the hardware samples the value of the index register and uses that for both the source and destination references. Meanwhile, the index register is modified. So it makes no sense to think about incrementing the register after the read operation but before the write operation or incrementing it twice.

It is not valid, however, to use a post-modify on the destination and an offset on the same index as source.

2.8.1.2 Mixing Indexed and Named Register Usage

Use of index registers does not result in any register allocation. Conceptually, this is similar to the C language behavior that “int *p;” does not allocate an integer.

Local memory can be allocated and managed manually. You can allocate and choose specific addresses for local memory use through #define’s; however, the preferred method is to use the memory allocation directives described in [Section 2.11.11, “Memory Allocation Directives”](#) to allocate blocks of local memory.

Caution: In the case of neighbor registers, the two methods (indexed and named) are conceptually exclusive. When the indexed method is being used, one would not be using the named method, and since the index defines a FIFO covering the entire register array, allocation is not relevant.

For transfer registers, however, indexed and named usage may be mixed. This is partially a result that I/O references work on named transfer registers and not indexed transfer registers. To cause the register allocation to occur, all of the registers need to have names and to be part of the .xfer_order instruction, regardless of whether the programmer will actually reference them by name. Additionally, the programmer needs to use .set or .use directives to indicate when these registers are being used.

2.8.1.3 Transfer Registers (xfer)

The xfer parameter specifies a Transfer register. Transfer registers are always specified with one or two \$ characters as a prefix. S-Transfer register use one \$ (example: \$xfer) while D-Transfer registers use two \$ (example: \$\$xfer). Read and write transfer registers are specified by how they are used. For example, reading \$xfer reads a S-Transfer Read register while writing the register writes an S-Transfer Write register

When an I/O instruction specify a reference count (ref_cnt) greater than 1, the data from multiple transfers are read or written from a contiguous set of Transfer registers. In this case the xfer parameter specifies the first register in the contiguous set. Since the instruction only specifies the first register in the contiguous set of registers, the assembler requires that the programmer indicate the names of registers that the programmer would like to use for the other contiguous registers. This is specified using the .xfer_order assembler directive.

2.8.2 Register Declarations

The main purpose of register declarations is to assist the programmer in catching bugs; primarily those resulting from typing the name of a register incorrectly. The use of register declarations may be made optional or required depending on the command-line switches. If declarations are not required, then a register that is used without being declared is implicitly declared with a global scope and an automatic lifetime.

2.8.2.1 Preferred Register Declaration Syntax

Registers are declared¹ using the `.reg` directive.

Registers can have four different attributes. These are specified as described later in this section by keywords. The default attribute values (i.e., the attributes specified with no keywords) are underlined. For a more thorough description, see the rest of this section.

Scope: This determines what part of the source code can reference this register. Conceptually, this can have one of three values (if the declaration occurs within a block, then block is the default scope; otherwise, module is the default):

- **Block:** The virtual register can only be referenced from its declaration to the end of the enclosing block. This is similar to a variable declared within the body of a C-function.
- **Module:** The assembler currently has no notion of “module”. This attribute is reserved for possible future use. At this release, the module attribute is effectively the same as having it at global scope except that the name is prefixed in the list file. This is similar to a top-level non-static variable in C.
- **Global:** The virtual register can be referenced from its declaration to the end of the module, or from within other modules (via extern declarations). This is similar to a top-level non-static variable in C.

Lifetime: This determines how the register allocator allocates this register. Conceptually, this can have one of two values:

- **Automatic:** The allocator will determine those parts of the code where the register contains a meaningful value. This is called the live-range of the virtual register. Two registers whose live-ranges do not overlap may be safely allocated to the same physical register. This is the normal situation.
- **Volatile:** The virtual register is allocated to a dedicated physical register; i.e. no other virtual register will be allocated to the same physical register. This guarantees that a reference to a different virtual register will never modify values in this virtual register. This would be needed if either the allocator’s algorithms compute the wrong live-range, or if the register were to be accessed asynchronously (e.g. a transfer register that was to be target of a reflector operation).

Direction: In the case of transfer registers, this indicates which of the read/write transfer registers are being allocated. This can take three values:

- **Read:** Only a read transfer register is declared.
- **Write:** Only a write transfer register is declared.
- **Both:** Both a read and a write transfer register (at the same address) is declared. This is needed for operations that do both a read and a write at the same time, e.g. test-and-set.

A transfer register that is not explicitly declared “read” or “write” (that is, it is implicitly declared as read/write or “both”) is considered as two separate but linked physical registers. That is, the live range is computed for each of the pair separately.

This means that declaring a transfer register with the “read” or “write” keywords has no effect on register allocation. This is because if it is declared as “both”, but only used as a “read”, then the write “half” of the register will have an empty live range, and so it won’t conflict with anything else. The same holds true if it is declared as “both” but only used as a “write” transfer register.

1. Note that index registers do not need to be declared. Since local memory can only be accessed by use of index registers, it is never declared.

The only advantage to declaring a register as "read" or "write" is that attempts to use that register incorrectly (for example, making a read transfer register the destination of an ALU instruction) results in an error. If the register were declared as "both", then such an attempt would not generate an error, although it might generate a warning.

When register declarations are not used, there is no way to mark a register as "read" or "write". In previous versions of the assembler, this was allowed via the `.xfer_order_rd` and `.xfer_order_wr` directives. This usage is obsolete and now generates a warning and is equivalent to `.xfer_order`". If you want the error checking previously provided by `.xfer_order_rd` and `.xfer_order_wr`, you must now declare the appropriate registers with either the `READ` or `WRITE` keywords.

Visibility: In the case of transfer registers, this indicates whether the register is visible to other microengines via the reflector. Neighbor registers are always visible. Conceptually, this can take one of two values:

- **Visible:** Other microengines can read/write this register.
- **Invisible:** Other microengines cannot read/write this register.

Note that it would be rare to have an absolute register declared with an automatic (non-volatile) lifetime. Since absolute registers are generally accessed by multiple contexts, it should generally have a volatile lifetime. Note that if an absolute register is declared with an automatic lifetime, the assembler may choose to treat it as if its lifetime were volatile.

For more details on visible/remote, see [Section 2.8.7](#).

A visible transfer register would automatically be considered volatile. The syntax of the register declaration is:

```
.reg [keywords]* name1 name2 ...
```

keywords: Zero or more keywords as described below.

names:: One or more register names. You cannot declare a register whose name matches one of the keywords.

The keywords define the attributes of the registers being declared, as defined by the following table:

Keyword	Meaning
volatile	If the volatile keyword is present, then the lifetime of the declared registers is set to volatile. Otherwise, the lifetime is automatic. Note that in some cases (e.g. named neighbor registers), the lifetime is always volatile, regardless of the absence or presence of this keyword.
global	If the global keyword is present, then the scope of the declared registers is set to global. Otherwise, if the declaration is within a block, the scope is set to block. If it is not within a block, then the scope is set to module.
visible	If the visible keyword is present, then the visibility of the declared transfer registers is set to visible. Neighbor registers are always visible.
read write	If either the read or write keywords are present, then the direction for transfer and neighbor registers is set accordingly. These attributes have no effect on GPRs. If neither keyword is given, or if both keywords are given, then the direction is set to both.

extern	If the extern keyword is present, then the named registers are declared elsewhere (either in this module or another that will be linked in). This is similar to the C-language construct "extern <i>type name</i> ".
remote	If the remote keyword is present, then the named transfer or neighbor registers must be declared in a different microengine and will presumably be the target of a neighbor write or a reflector reference. These are resolved by UCLD. The remote register must be declared as visible in the remote microengine in order to be seen by this microengine.

Usage of these keywords is summarized in the following tables:

	GPR	Xfer	Neighbor
volatile	valid	valid	implied ^a
global	valid	valid	implied
visible	error	valid	implied
read/write	error	valid	error
extern	valid	valid	valid
remote	error	valid	valid

- a. **Implied** means that it may be specified or not. In either case the program behaves as if it were specified

Keyword compatibility				
	volatile global visible	read/write	extern	remote
volatile global visible	X	OK	error	error
read/write	OK	X	OK	error
extern	error	OK	X	error
remote	error	error	error	X

Rules:

1. Remote cannot be used with any other keywords.
2. Extern can only be used with read or write.
3. Other keywords may be freely mixed.

It is valid to declare the same register as ".reg extern" and ".reg global". It is also valid to declare a remote register multiple times. This would occur when a macro which contains a remote register declaration, is used multiple times.

It is valid to declare the same register name as remote and non-remote. In this case, context will determine which register is referenced. This usage is confusing and should be avoided, but there may be strange cases where this is required (such as two microengines running identical code and which want to access each other's transfer registers).

To declare registers with a “module” scope, they should be declared outside of any `.begin/.end` blocks and without the `GLOBAL` keyword. (See [Section 2.8.1.2, “Mixing Indexed and Named Register Usage”](#).)

The attribute implications are summarized as follows:

```
neighbor ⇒ global, volatile, visible
visible  ⇒ global, volatile
remote   ⇒ global
extern   ⇒ global
```

The response to declaring a register with the same name as a previously declared register is summarized in the following table. Note that for the “first register/Block” column, the assumption is that the second register is declared within the same block.

		First Register				
		Global	Module	Block	Extern	Remote
Second Register	Global	Error	Error	Warn	OK ¹	OK
	Module	Error	Error	N/A ²	Error	OK
	Block	Warn ³	Warn ³	Error	Warn ³	OK
	Extern	OK ¹	Error	Warn	OK ¹	OK
	Remote	OK	OK	OK	OK	OK

¹First and second registers refer to same register.

²You can't declare a module-scoped register within a block.

³Generates a high-level (level-4) warning.

In such cases, more than one register will exist with the same name. Which is referenced is a function of the code. The following example shows that a local register declaration will mask a global variable declaration of the same name within the scope of the local block:

```
.begin
.reg foo          // defines a local variable named foo
.reg global foo   // defines a global variable named foo
foo...           // reference to foo is to the local variable
.end
foo ...          // reference to foo is to global variable
```

The directive `.xfer_order` does not declare registers. The arguments to `.xfer_order` need to be declared before they are used in the `.xfer_order`. If programmers want to avoid writing out the variable list twice, they can use a macro similar to:

```
#macro reg_order[type, regs]
```

```
.reg type regs
.xfer_order regs
#endm
reg_order[global volatile, $1 $2 $3]
reg_order[, $4 $5 $6]; Note, the null type field
```

For compatibility with earlier releases, `.xfer_order_rd` and `.xfer_order_wr` are still accepted; however, they behave the same as `.xfer_order`. In order to get the checking that was previously implied by these directives, the registers need to be declared with either the read or write keywords.

2.8.2.2 Details of Volatile and Visible

A register that is declared volatile and global has the same meaning as volatile in previous versions of the assembler. That is, such virtual registers are allocated to dedicated physical registers. However, registers can now be declared as volatile without being declared global. This means that the virtual register is allocated to a dedicated physical register within the defining block. If the code flow leaves that block, then the register could be reallocated. The typical use for this feature would be microcode where different code blocks corresponded to different independent threads. In this case, the defining block for the volatile registers would consist of all of the code for a given thread. For example, the fact that context-0 had a particular volatile register should not affect the allocation of the registers local to context-1.

Since volatile no longer implies global, it is also possible to have non-global visible registers. In this case, there is the added restriction that any particular ME could only declare one visible register with a given name. If two were declared, then there would be no way to distinguish them. For example, the following code would be invalid:

```
.begin
.reg visible foo
...
.end
.begin
.reg visible foo // Not allowed because there are two visible foo's
```

Note that this would have been legal if either or both declarations had omitted the visible keyword.

2.8.2.3 Compatible Register Declaration Syntax

For compatibility with existing code, there is an alternate syntax for declarations.

The directive:

```
.local reg1 reg2 reg3 ...
```

is defined to be functionally equivalent to

```
.begin
.reg reg1 reg2 reg3 ...
```

Similarly, the directive:

```
.endlocal
```

is defined to be functionally equivalent to

```
.end
```

Saying “functionally equivalent” means that it “behaves the same as”. However they are not fully equivalent because the compatible and new syntax cannot be freely mixed. That is, a “.local” needs to be closed with a “.endlocal”. The following two cases, for example, would be illegal:

```
.local
...
.end // Error: should be .endlocal
```

or

```
.begin
...
// Error: should be .end
```

2.8.2.4 Dealing with self-write neighbor regs

There is an ambiguity when writing to named neighbor registers. Normally, doing so writes to a register in the neighboring ME. However, a CSR bit can be set that causes writes to go to the same ME. This might be used, for example, to store certain constants or pseudo-constants for later use.

In general, the assembler cannot determine the setting of this CSR bit, so it needs the programmer to indicate whether a write to a named neighbor register is going to the neighbor ME or to the self ME.

The normal case is pretty straightforward. If the destination register is declared via a `.reg` directive, then it is assumed to be local to this ME. If it is declared via a `.reg remote` directive, then it is assumed to be in the neighbor. The one ambiguous situation is if the register is declared with both a `.reg` and a `.reg remote` directive. In this case, it will be assumed that the actual destination of the write is the register in the remote ME. While this usage may be needed in rare occasions, it is confusing and should be avoided.

Note that it is up to the programmer to ensure that the destination of the neighbor write matches the current setting of the CSR bit. If these do not match, then a random register in the wrong ME will be modified.

2.8.3 Aggregate and Array Support

2.8.3.1 Register Arrays

The assembler supports the notion of an array of registers. This is called an "aggregate". These are declared by giving a bracketed size following the name in ".reg". For example:

```
.reg $x[3]
```

declares an aggregate called "\$x" consisting of three registers. The maximum size for an aggregate is 128 registers. In the case of remote registers, the size can be left off, e.g. ".reg remote \$r[]". You cannot have an aggregate with the same name as a non-aggregate. So the following would be invalid:

```
.reg $a[3]
.reg $a ; illegal because of $a[3] above
```

Aggregates cannot be implicitly declared, they must be declared explicitly.

The .xfer_order directive takes entire aggregates, not aggregate elements. For example:

```
.xfer_order $a $x $b
.xfer_order $a $x[0] $b ;; ERROR, can't use indices
```

This would result in the registers being ordered as "\$a \$x[0] \$x[1] \$x[2] \$b".

In instructions and most directives (exclusive of .reg and .xfer_order), elements of an array are referenced with a bracketed index. Continuing the example above:

```
immed[$x[0], 1]
alu[$x[1], $x[2], +, 1]
alu[--,--,b,$x] ;; ERROR: missing index
immed[$x[3], 0] ;; ERROR: index out of range
```

2.8.3.2 Compatibility with Earlier Releases

In previous releases, brackets were not allowed in directives and were ignored in instructions. This resulted in the odd code style:

```
.reg $y0 $y1 $y2
.xfer_order $y0 $y1 $y2
immed[$y[0], 0]
...
```

Code that does this should be rewritten to use the new style (as described in the previous section). For the current release, to maintain some compatibility with earlier code and to give programmers a chance to update their code, the will exhibit the following behavior:

- If a reference looks like an aggregate element reference, but there is no register with the specified name, then the assembler will remove the brackets and try to find that register. It will also generate a warning that the code should be updated. For example, if there was a reference "\$name[3]" and there was no register named "\$name", UCA would look for "\$name3".
- If a reference looks like an old-style aggregate (e.g. \$name0), but there is no register with the specified name, then the assembler will look up the register as a true aggregate (e.g. \$name[0]). It will also generate a warning that the code should be updated. For example, "\$name0" will match a register that was declared as ".reg. \$name[4]".

This behavior is a temporary one to give programmers more of a chance to update their code. Future releases will drop this behavior.

2.8.3.3 Doubled Signal References

A doubled signal is another type of "aggregate". For DRAM references, both signals are needed to indicate that the I/O has completed. For other I/O instructions that generate a doubled signal, one "half" indicates that the write-transfer-register has been consumed, and the other "half" that the read-transfer-register has been filled.

From a syntactic point of view, references to a doubled signal name by itself (e.g. "sig") will refer to essentially the entire pair. To reference the low half, one would append either "[0]" or "[write]" to the signal (e.g. "sig[0]" or "sig[write]"). To reference the high half, one would append either "[1]" or "[read]" to the signal (e.g. "sig[1]" or "sig[read]"). References to "[0]" versus "[write]" (or "[1]" versus "[read]") are equivalent; i.e. either can be used interchangeably. The suggested use is that when the signal refers to a DRAM transaction "[0]" and "[1]" would be used, and when the signal refers to another transaction, "[write]" and "[read]" would be used.

Note that doubled signals look very similar to an array of signals (of length 2), except that in this context, "write" is a synonym for "0", and "read" is a synonym for "1" (as an aid in readability).

Doubled signals can be referenced three ways: via the address operator, via a `br_*signal`, and via a `ctx_arb`.

The address operator can be applied to the unqualified name (e.g. "sig"), or to the qualified ones ("sig[0]" or "sig[read]"). The address of "X" would be the address of "X[0]". Examples would include:

```
immed[x, &s]           ; references low/write half
immed[y, (1+&remote(s))] ; references high/read half
```

The `br_*signal` (`br_signal`, `br_!signal`) instructions always reference qualified names. So the following would be valid:

```
11#: br_!signal[dram_sig[0], 11#]      ; references low half
12#: br_!signal[dram_sig[1], 12#]      ; references high half
13#: br_!signal[sram_sig[write], 13#]   ; references low half
14#: br_!signal[sram_sig[read], 14#]    ; references high half
```

The `br_*signal` functions should not reference the unqualified names (see Section 4.1.6.3.2).

The `ctx_arb` instruction can reference qualified or unqualified names. If the unqualified name is used, then it will be automatically doubled by the assembler, for example:

```
dram[...], sig_done[s]
ctx_arb[s]           ; equivalent to ctx_arb[s[0], s[1]]
```

For advanced users, the individual halves of the signal can be referenced by using the qualified names, for example:

```
sram[swap, $x, addr,0], sig_done[s]
ctx_arb[s[read]] ; not automatically doubled due to "[read]"
...; it is now safe to access $x as a read xfer reg
ctx_arb[s[write]] ; not automatically doubled due to "[write]"
...; it is now safe to access $x as a write xfer reg
; or to reuse "s"
```

Note that this is advanced usage. In most cases, the programmer would use the unqualified name. It is only in extremely rare circumstances that advanced programmers would use qualified names in a `ctx_arb`.

These points are summarized in the following table:

Reference	Uses		
	Address of	br_*signal	ctx_arb
s (unqualified)	OK	Invalid	OK (automatically doubled)
s[0] (qualified, DRAM) s[write] (qualified, non-DRAM)	OK	OK	OK (not doubled)
s[1] (qualified, DRAM) s[read] (qualified, non-DRAM)	OK	OK	OK (not doubled)

Note that if a doubled signal is half-consumed before a `ctx_arb` (either through a `br_*signal` or `ctx_arb` with a qualified name), then the signal must be referenced with a qualified name. For example:

```
sram[swap, $x, ...], sig_done[d]
ctx_arb[d[read]]
ctx_arb[d] ;; ERROR: must be d[write] or d[0]
```

2.8.3.4 Usage Notes

In typical usage, programmers won't need to use `br_*signal` for I/O references, so they can just use unqualified names in the `ctx_arb` instruction, and virtually all aspects of doubled signals are handled automatically by the assembler. I.e. when using `ctx_arb` and unqualified signal names, programmers can treat doubled and non-doubled signals in the same manner.

In some cases, which may be rare, programmers may want to poll a doubled signal using `br_!signal`. In this case, they need to be aware of whether the signal is doubled or not, and if it is, they need to use the appropriate qualified name.

In very rare cases, for advanced programmers, they may want to `ctx_arb` on a qualified doubled-signal name, so that (typically) they can access the read transfer registers before the write completes. This is an advanced technique to be used in special circumstances. It is not expected to be used by the majority of programmers.

2.8.3.5 Compatibility Issues

In previous releases, in the `br_*signal` usage, a different syntax was used:

Previous Syntax	Current Syntax
<code>s</code>	<code>s[0]</code> or <code>s[write]</code>
<code>s+1</code>	<code>s[1]</code> or <code>s[read]</code>

For compatibility with earlier releases, `br_*signal` will continue to accept "s" rather than "s[0]" or "s[write]", although such use should be discouraged. This is to avoid confusion with the automatic doubled of `ctx_arb`. Similarly, the syntax "s+1" will be accepted for "s[1]", although this is not recommended. At some future release, such forms may cease to be accepted.

Similarly, where one used to write:

```
.if (signal(s+1))
```

one should now write:

```
.if (signal(s[1]))
```

2.8.4 Transfer Order(.xfer_order)

The `.xfer_order` directive describes an ordering of transfer registers. In the case of transfer requiring multiple 32-bit data transfers, it is necessary to describe to the assembler those 32-bit register names that must be contiguously ordered in the transfer register address space. This is so that the intended data can be accessed predictably by symbolic specification to individual registers. The `.xfer_order` reserved name is followed by the ordered list of register names (order increases from left to right).

Instruction Format

```
.xfer_order reg1 reg2 ...
```

For compatibility with earlier releases, `.xfer_order_rd` and `.xfer_order_wr` are still accepted; however, they behave the same as `.xfer_order`. In order to get the checking that was previously implied by these directives, the registers need to be declared with either the read or write keywords.

All registers that are related via `.xfer_order` must have the same scope. So, for example, the following would be invalid:

Examples: Incorrect Usage of Xfer_order (1)

```
.begin
  .reg $x1 $x2
  .begin
  .reg $x3 $x4
  .xfer_order $x1 $x2 $x3 $x4 // invalid
```

Similarly, the following would also be invalid:

Examples: Incorrect Usage of Xfer_order (2)

```
.reg $x1
.reg visible $x2
.xfer_order $x1 $x2 // invalid
```

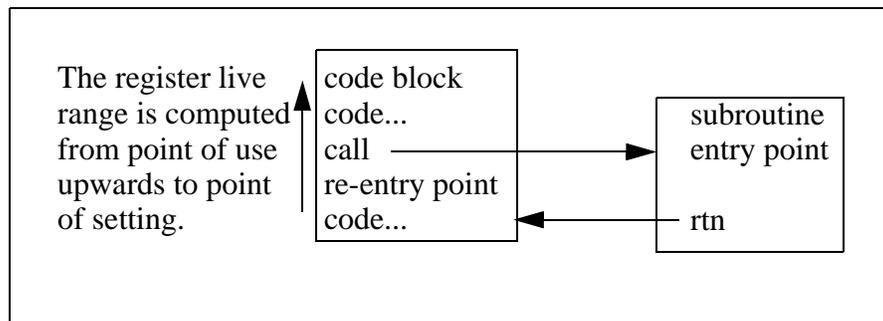
This is because \$x1 is declared with a module scope, and \$x2 is declared with a global scope (since visible implies global).

2.8.5 Register Lifetime Details

For non-volatile registers, the lifetime is computed automatically. Basically, the register is considered “live” everywhere it is used as a source of an operation. This “liveness” is then propagated backwards, up the flow graph until it is terminated by an operation that “sets” or assigns a value to that register.

If the register is not volatile and has a block scope, then the live-range is truncated when it leaves the register’s defining block. The exception to this is when it makes a subroutine call/return.

More particularly, when going up the flow graph, the live range is truncated when the current microword is inside the register’s defining block, the next microword in the graph is not, and the branch was not caused by a RTN. This is illustrated in the figure below. If the current microword is the one labeled “re-entry point”, then the next-microword is not in the **code block**, but since the next-microword is a RTN, the live range is not truncated and continues up through the subroutine and back into the code block.



A similar situation exists when extending the live range of a transfer register. (It is extended going down the flow graph from the I/O operation to the completion of the I/O operation, typically a `ctx_arb`.) In this case (going down the flow graph), the live range is truncated when the current microword is in the register's defining block, the next microword is not, and the next microword is either not in a subroutine or is in the same subroutine as the current microword. That is, a "subroutine call" is defined as a branch into a subroutine block from someplace not in that subroutine block, and the live range going down the flow graph is truncated when it leaves the defining block, unless it is a "subroutine call" as defined above.

Consider the case of a block-scoped register that wants to maintain its value outside of its block. This would be similar to a variable in C defined within a function with the static qualifier. This could be done in one of two ways:

The register could be declared as volatile. This will guarantee that no one else will clobber that register, although it will "use up" a dedicated physical register.

The register could be declared as having a module or global scope. This makes it potentially visible elsewhere in the code, but would result in its lifetime not being truncated.

2.8.5.1 MEv2 Queue Information

The MEv2 I/O operations (with the exception noted below) are inherently unordered with respect to each other. That is, the implementation may enforce some particular order, but the architecture does not. As a result, all MEv2 operations go to the "unknown" queue, and a signal must be specified on each operation. You may take advantage of the ordering of a particular implementation as determined from other sources (and indicated to the assembler through the mechanisms described in the section entitled "[Determining when I/O operations complete](#)" on page 66), but such code will be inherently non-portable.

The exception to the above is operations to the MSF with the ordered optional token (see [Table 2-10](#)). The ordering of MSF transactions is limited in certain circumstances (e.g. between writes to the TBUF and TX_VALIDATE). Thus code that is designed to be portable with future versions of the architecture should limit use of the ordered token to this case. The current implementations, however, maintain order of all transactions. If you want to take advantage of this, you may use the ordered token in more situations. The danger in doing so, however, is that such code may break for future generations of the chip.

Table 2-10. MEv2 Logical Queues

Instruction	Condition	Queue
MSF	read, ordered	MSF-Read
	write, ordered	MSF-Write
	other	unknown
other		unknown

2.8.6 Signal Declarations

Signals are declared in a manner similar to registers:

```
.sig [keywords]* name1 name2 ...
```

keywords: Zero or more keywords as described below.
namen: One or more signal names. You cannot declare a signal whose name matches one of the keywords.

The keywords define the attributes of the signals being declared, as defined by the following table:

Keyword	Meaning
volatile	If the volatile keyword is present, then the lifetime of the declared signals is set to volatile. Otherwise, the lifetime is automatic.
global	If the global keyword is present, then the scope of the declared signals is set to global. Otherwise, if the declaration is within a block, the scope is set to block. If it is not within a block, then the scope is set to module.
visible	If the visible keyword is present, then the visibility of the declared signals is set to visible.
extern	If the extern keyword is present, then the named signal needs to be defined elsewhere (either in this module or another that will be linked in).
remote	If the remote keyword is present, then the named signals must be defined in a different microengine and will presumably be the target of a remote reference (e.g. sending an inter-thread signal). These are resolved by UCLD. The remote signal must be declared as visible in the remote microengine in order to be seen by this microengine.

The namespace for signals is the same as for GPRs, i.e. an alphabetic character followed by zero or more alphanumeric characters. Most particularly, there is no type prefix. This is because there is no reasonable scenario where a macro argument could represent a register or a signal, and the macro has to determine which it is.

The **volatile** keyword would in general be needed if the signal were being generated other than in response to some action within this thread. This might be needed, for example, for an inter-thread signal. Preferred Register Declaration Syntax

The keyword restrictions are the same as for registers as described in [Section 2.8.2.1, “Preferred Register Declaration Syntax”](#).

When a signal is consumed via a `ctx_arb`, all of the signals will be checked for number. For any of these signals, if the source along any path generates a doubled signal¹, then all sources must generate a doubled signal. Each doubled signal will implicitly include the next higher signal in the `ctx_arb`. For more information on doubled signals, refer to [Section 2.8.3.3, “Doubled Signal References”](#).

2.8.7 Use of REMOTE Keyword

Registers are defined by the code for the micro-engine in which they are located. Other micro-engines (e.g. those doing a neighbor-write or reflector operation) reference these via the “remote” keyword.

1. Some I/O operations generate two signals; i.e. the signal specified for the I/O operation must be even, and that signal and the next higher odd signal is also returned. Such a signal is called a “doubled signal” in this document. For more information on double signals, refer to [Section 3.1.2.4, “Event Signals”](#)

Similarly, signals are defined by the user or recipient of the signal (e.g. the micro-engine doing the `ctx_arb`). Other micro-engines that wish to send a signal (e.g. via a CSR write) would declare these signals with the “remote” keyword. Note that the only valid operation on a remote signal is taking the address of it. It is never valid to `ctx_arb` on a remote signal, since that signal does not exist in the current micro-engine.

Note also that in order to successfully resolve the remote register or signal, it must be declared as “visible” in the remote micro-engine.

This is illustrated in the following examples:

Examples: ME0 does a named neighbor write to ME1

ME0 code	ME1 code
<code>.reg remote n\$name</code> <code>alu[n\$name, ...]</code>	<code>.reg visible n\$name</code> <code>alu[xxx, --, b, n\$name]</code>

Note in this example that neighbor registers are located in the ME1 that reads them and not in the ME0 that writes them.

Examples: ME1 does a reflector read from ME3

ME1 code	ME3 code
<code>.reg remote \$name</code> <code>.reg \$my_xfer</code> <code>cap[read, \$my_xfer, 3, \$name, 0, 1], ctx_swap[sig]</code>	<code>.reg visible \$name</code> <code>immed[\$name, ...]</code>

Note in this example that `$name` is located in ME3, and it is accessed using normal source/destination references. In ME1, it is declared as “remote” and referenced via the “cap” command.

Examples: ME4 sends a signal to ME2 (thread 1)

ME4 code	ME2 code
<code>.sig remote rsig</code> <code>cap[fast_wr, ((2<<7) (1<<4) (&remote(rsig,2))), interthread_sig]</code>	<code>.sig visible rsig</code> <code>ctx_arb[rsig]</code>

Note in this example that the signal is declared and used normally in ME2 (except that it is declared “visible”), and that it is referenced via the address operator in ME4.

2.8.8 Address Operator

There is a need to be able to take the address of transfer registers and signals (the “address” of a signal is the signal-number associated with it). This is needed for registers and signals defined locally (i.e. non-remotely) and for ones declared remote.

Note that the address of a register is a "long-word address" (or equivalently a "register number"). In other words, all low-order bits of the address are significant, and the addresses of the first few registers are 0, 1, 2, etc. (not 0, 4, 8). The "address" of a signal is its signal number, in the range of 1...15.

When registers and signals are defined locally, this is done by prepending an "&" to the register/signal name; e.g.: `&$xfer`, `&sig_name`.

When registers and signals are declared remotely, the reference is defined with the following pseudo-function:

```
&remote(name, ME_num, ...)
```

name: Name of remote register/signal
ME_num: Number of remote microengine

In the case where a register/signal is declared in a different block (typically corresponding to a different context), it is also considered "remote". It is referenced with the "&remote(name)" construct as described above, but with no ME_num specified.

If more than one ME_num is specified, then a check will be made to make sure that the specified register/signal has the same address in all of the listed microengines. This can be used, for example, when computing the address of a transfer register for a cap command where the microengine to be reflected to is being selected at run-time. For example:

```
; Assume ME is in range of 1...3
immed[reg1, ((1<<15) | (CTX<<6) | (&remote($r,1,2,3)<<2))]
alu[reg1,reg1,or,ME,<<10)
cap[read, $x, regA, TMP, 1], ...
```

In all of these cases, the reference is usable where a constant would be used. Depending upon the usage, registers or signals that have their address taken may or may not need to be declared volatile by the programmer.

These operators (as well as imported variables) can also be used in constant expressions.

In the context of constant expressions, the address of a remote (or local) neighbor register can be taken. Note that when taking the address of a remote neighbor register, the ME number of the neighbor does not need to be specified, and so the &remote() function is not needed. That is, to take the address of a remote neighbor register, it is sufficient to say "&n\$reg".

Note also that the address of a local neighbor register can be taken within a constant expression but not outside of a constant expression. Constant expressions have enclosing "()", so the following is valid:

```
.reg n$local gpr
immed[gpr, (&n$local)]
```

but the following is not:

```
.reg n$local gpr
immed[gpr, &n$local] ; Error: must be within constant expr
```

2.8.8.1 Accumulating Results for ctx_arb[--]

In order to use the ctx_arb[--] feature, you need to be able to accumulate a subset of certain signals into a register. In order to do this, you need a way to get the size of the signal and way to shift it to the correct position. To support this, there is a built-in function for constant expressions

```
mask(sig)
```

which will expand to "1" for normal signals and "3" for doubled signals. This would typically be used to generate a bit-mask to be written to active_ctx_wakeup_events as illustrated in the example below.

The mask() function behaves differently based on whether the specified signal is active at the time of use (e.g. if the use of mask() is between the I/O and the ctx_arb). If the signal is active, the value of the mask function will be based on whether the active signal is doubled or not. This is because another, unrelated use of the signal may use it in the other sense. However, if the signal is not active at the time that mask() is used, but the signal is only used in a doubled or single sense, the mask() function succeeds. If the signal is not active where mask() is used, and the signal is used in both a single and doubled manner, then an error results.

Additionally, the alu (alu_shf) instruction will support syntax for shifting based on the address of a signal. In particular, the shift token can take any of the following forms:

```
<<&sig_or_reg
<<&remote(sig_or_reg)
<<&remote(sig_or_reg, me)
<<(constant_expression)
```

This would be used as indicated in the following example (for an explanation of ".io_completed", see the section entitled "[Determining when I/O operations complete](#)" on page 66"):

```
alu[sig_mask, --, b, (mask(sig1)), <<&sig1]
.if (...)
alu[sig_mask, sig_mask, or, (mask(sig2)), <<&sig2]
.endif
.if (...)
alu[sig_mask, sig_mask, or, (mask(sig3)), <<&sig3]
.endif
local_csr_wr[active_ctx_wakeup_events, sig_mask]
ctx_arb[--]
.io_completed sig1 sig2 sig3
```

Note that if the mask() function is invoked where a signal is live and single, but then that value is later used when the signal is doubled, (or vice versa), the code will fail with no warnings or errors. For example:

Example of incorrect usage of mask()

```
sram[read, $x, a,0, 1], sig_done[sig1] ; sig1 is single
alu[sig1_mask, --, b, (mask(sig1)), <<&sig1] ; mask() is 1
ctx_arb[sig1]
...
.if (...)
sram[swap, $x, a,0], sig_done[sig1] ; sig1 is now doubled
alu[tot_mask, tot_mask, or, sig1_mask] ; sig1_mask is 1
```

```

; ERROR: tot_mask is now incorrect. It is 1, but should be 3
.endif
local_csr_wr[active_ctx_wakeup_events, tot_mask]
ctx_arb[--]
.io_completed sig1 ...

```

2.8.8.2 Examples of Address Operator and Visible/Volatile Signals

The first example is one context (context-0) signaling another (context-1) in the same ME:

```

.if (ctx() == 0)
  .sig remote s
  local_csr_wr[..., (&remote(s)... )
...
.elif (ctx() == 1)
  .begin // begin block for context-1
    .sig visible s
    .while(1)
    ...
  .endw
  .end
.elif ...

```

In this case, context-0 is using a remote declaration and the "&remote(name)" construct to reference a signal declared within the same ME, but which is local to a different block.

Note that if context-1 was in a different ME, then the picture would look almost identical, except that the reference would be "&remote(name,ME_num)".

In either case, there is a problem if the programmer wants to use the same visible signal in different, independent contexts. There are several ways in which this could be addressed:

1. Use three different names, e.g. "s1", "s2", etc.
2. Use a .begin/.end block that included multiple contexts (but preferably not the unnecessary contexts, as this would defeat the purpose of making the visible signal non-volatile).
3. Make the signal "s" volatile and global, and just live with wasting a signal in the other contexts.

Another example where this mechanism would be useful is in the case where one ME (the master) is initializing some data structure and wants to hold off the other MEs (the slaves) until the structure is initialized. A typical way to do this is to have the master ME send an inter-thread signal to the slave MEs. This signal would have to be visible, but it would only be used during initialization. This could be handled in slaves as:

```

.if (ctx() == 0)
  .begin
  .sig visible wakeup
  lab#: br_!signal[wakeup, lab#]
  .end
.endif

```

In this case, after the begin/end block was exited, the physical signal allocated to "wakeup" would be available for reuse.

The master would do something like:

```
.reg remote wakeup
cap[fast_wr, ((0 << 7) | &remote(wakeup,0)), interthread_sig]
cap[fast_wr, ((1 << 7) | &remote(wakeup,1)), interthread_sig]
cap[fast_wr, ((2 << 7) | &remote(wakeup,2)), interthread_sig]
...
```

2.8.9 Signal Lifetime Details

Normally, the lifetime of a signal extends from the I/O operation that generates the signal to the point where the signal is “consumed” via a context arbing operation or a br_signal operation. However, there are two cases where the signal is not generated in response to an I/O operation.

The first is where the generator of the signal is asynchronous with respect to the microengine in question. An example of this is an inter-thread signal. In this case, it should probably be declared as volatile.

The second case is where the microengine in question initiates the action that will eventually result in a signal, but that action is not an I/O operation. An example of this might be a CSR write which initiates a PCI DMA operation. In this case, a .set directive can be placed following the I/O instruction that writes to the PCI register that initiates the PCI DMA. This tells the assembler that the signal is now active.

2.8.10 Register Allocator Directives

There are several areas where limitations in the assembler cause it to make overly pessimistic assumptions. There are two ways that this can be dealt with:

- The simple approach is to do nothing. The assembler should always “do the right thing” or do the safe thing. The only two problems are that there may be excessive warnings and the register usage may not be optimal. The problems with the warnings can be handled via the warning mechanisms. If the register usage is a problem, then the following approach can be used.
- There is a series of directives (as described in this section) that allows the advanced user to more carefully tune the register allocation process and possibly achieve better register utilization. The downside is that use of these directives will be less-obvious to a casual user and would require a greater understanding of the register allocation process.

There are four areas where limitations of the register allocator may cause spurious warnings or non-optimal register allocation. These are described in the following four sections.

Register Used Before Being Set

If there is no path that sets a register before using it, then a low-level warning is issued (“Used before set”). If there were some paths that set it and some that do not, then a high-level warning is issued (“Maybe used before set”).

A typical example of this second kind would be:

```
.if (condition_1)
    immed[reg, 0]
.endif
...
```

```
.if (condition_1)
    alu[... ,reg]
.endif
```

The assembler does not know that the second conditional is taken only if the first is already taken. It assumes that the first conditional can be skipped and that the second can be taken. This results in the register being used before being set.

To address this in the case that the warning level is set to include this warning and the programmer has verified that there actually is no problem (i.e. in the above example, if the two conditions were different, there would be a real problem), the programmer could use the `.set` directive.

```
.set reg1 reg2 ...
reg1 reg2 ...:One or more registers to be "set".
```

This directive looks like an assignment to the register allocator (thus getting rid of the “used before set” warning), but it does not generate actual code.

Correct behavior can always be achieved by placing the `.set` directive at the start of the affected block. More optimized usage would place the directive immediately before the conditional causing the problem. Using the above example, the following would always be valid:

```
.begin
.reg reg
.set reg          // .set directive at start of block
...
.if (condition_1)
    immed[reg, 0]
.endif
...
.if (condition_1)
    alu[... ,reg]
.endif
```

But the allocator may use the registers more effectively if it is placed just before the conditional:

```
.begin
.reg reg
...
.set reg          // .set directive before conditional
.if (condition_1)
    immed[reg, 0]
.endif
...
.if (condition_1)
    alu[... ,reg]
.endif
```

Note that it would be an error to place it after that conditional (i.e. between the two conditionals). In this case, you would be “assigning” a value to the register after having done so with an actual assignment. In this case, it would appear to the register allocator that the register was set at the `.set` directive and then used, and hence the setting of the register using the `immed` instruction was irrelevant (i.e. that value was never used). It would therefore be free to clobber the value of `reg` between the `immed` and the `.set` directive.

This directive would also be required if the setting of transfer registers were done using the index register. In this case, the assembler would not know which registers were actually being set. Presumably the programmer knows and can use the `.set` directive to tell the assembler.

Note also that this directive should not be placed after register declarations as a matter of course, because this may mask real bugs. This directive should only be used after the programmer has determined that the warning is caused by a limitation of the assembler and not due to a potential problem with the code.

A similar directive, `.set_sig` can be used with signals:

```
.set_sig sig1 sig2 ...
sig1 sig2 ...: One or more signals to be "set".
```

This would be used in the case where the signal is being generated other than by an I/O operation. An example of this would be when writing to the CSR that initiates a PCI DMA operation.

Refer to the section [“Use of `.set` and `.use` with transfer registers” on page 68](#) for additional information on using the `.set` directive.

Determining when I/O operations complete

The assembler has to determine when I/O operations, particularly write operations complete. As detailed in [Section 2.8.13.1, “Transfer Register Lifetimes”](#), the assembler attempts to determine when a write I/O operation completes, but it may not be able to do so in all circumstances. If it cannot, it will report an error.

To indicate that an I/O operation is completed, the programmer would use one of the `.io_completed` directives:

```
.io_completed name1 name2 ...
.io_completed_type type

name1 name2 ...: One or more signals or transfer registers whose I/O operations
                have been completed.
type:           Memory type (listed below)
                SRAM, DRAM, SCRATCH, MSF, CAP, HASH, PCI
```

The directive ends the live ranges of referenced I/O operations. An operation is referenced if at the directive it is not completed and any of:

- The directive lists a signal used by the operation (or on a following operation using the same queue).
- The directive names the type of the operation.
- The directive lists any of the registers involved with the operation.

Ending the operation ends the live ranges of all parts of the referenced operation (e.g. the transfer registers and the signal)¹.

One common case where the programmer needs to indicate where an I/O operation is completed is the same scenario as described in [“Register Used Before Being Set” on page 64](#).

For example, consider:

```
.if (condition_1)
```

```

        sram[write, $xfer, ...], sig_done[sig]
    .endif
...
    .if (condition_1)
        ctx_arb[sig]
    .endif

```

The assembler doesn't know that after the I/O operation has been issued, the `ctx_arb` must be executed. It assumes that the `ctx_arb` could be skipped. The solution would be to place after this code:

```
.io_completed sig
```

Note also that there would have to be a `“set_sig sig”` before the first conditional. Otherwise, you would get a warning that the signal might be used before it was set.

Another example is that while the MEv2 spec does not guarantee ordering between certain I/O operations, the user may know (and for some reason want to take advantage of) some implementation guaranteeing the order. In this case, the “correct” way to handle it is to use a different signal on all of the operations, and then to context-`arb` on all of them, but for some reason, the programmer may not want to do this. For example:

```

(1) sram[write, $x1, ...] ; no signals specified, non-portable
(2) ...
(3) sram[write, $x2, ...], ctx_swap[...]

```

To indicate to the assembler that use of the register is finished, the programmer could use:

```

(1) sram[write, $x1, ...] ; no signals specified, non-portable
(2) ...
(3) sram[write, $x2, ...], ctx_swap[...]
(4) .io_completed $x1

```

The lifetime of `$x1` would extend from the setting of the register, through the I/O operation, and continue to an appropriate `.io_completed`. Note that in this example, the user could also have used `“io_completed_type sram”`. Note also that this directive is not needed if the assembler can determine when the use is completed.

Note that since GPRs cannot be used with `.io_completed`, there is no ambiguity as to whether the name refers to a signal or to a register.

Using registers indirectly

A similar but slightly different problem occurs when read transfer registers are referenced via the index register. The issue is that the assembler doesn't know which registers are addressed by the index register, so it doesn't know when the read values are actually used.

```

1. Note that this can be abused with the unobvious construct:
    .reg $x $y
    .xfer_order $x $y
    sram[read, $x, a, 0, 2], sig_done[s]
    ...
    .io_completed $y

```

Such usage is confusing and should be avoided.

The programmer needs to tell the assembler that the transfer registers in question are being used up to the point when they are not. This can be done with the `.use` directive:

```
.use reg1 reg2 ...
reg1 reg2 ...: One or more registers whose use has been completed.
```

Similar to the `.set` directive, this appears as a use of the named registers but without generating any microwords.

Examples: indirect usage

```
.xfer_order $1 $2 $3
sram[read, $1, addr1, addr2, 3], ctx_swap[sig]
// set up transfer register index register (not shown)
alu[... , *$index++] // uses $1
alu[... , *$index++] // uses $2
alu[... , *$index++] // uses $3
.use $1 $2 $3
```

Since those three registers are “used” by the `.use` directive, the lifetime of those registers will extend from the I/O operation to the `.use` directive, encompassing the uses from the index register.

Refer to the next section for additional information on using the `.use` directive with transfer registers.

Use of `.set` and `.use` with transfer registers

The `.set` directive can be used with both “read” and “write” transfer registers, but `.use` can only be used with “read” transfer registers (but `.use_wr` can be used with “write” transfer registers, see below for details).

More particularly, if `.set` is applied to a R/W (both) transfer register, it is considered as “setting” both the read register and the write register. If `.use` is applied to a R/W (both) transfer register, it is considered as a use only of the read register. If `.use` is applied to a register declared as “.reg write”, then it generates an error.

Variants on `.set` and `.use` exist:

```
.set_rd reg1 reg2 ...
.set_wr reg1 reg2 ...
.use_rd reg1 reg2 ...
.use_wr reg1 reg2 ...
```

These variants can only be used with transfer registers of the appropriate type (e.g. “.set_rd” cannot be used with registers that are not transfer registers or which have been declared as “write”). An example of when you would use these is:

```
sram[read, $X, a,0, 1], ctx_arb[s] ; reads (sets) $X.read
... ; area-1
...
... ; set t_index to point to $X
immed[*$index, 0] ; sets $X.write
.set_wr $X
sram[write, $X, a,0, 1], ctx_arb[s] ; writes (uses) $X.write
...
```

```
alu[--,--,b,$X] ; use $X.read
```

The problem is that if one uses ".set" rather than ".set_wr", then it will "set" both the read and write transfer registers, so it will appear that the read of \$x is useless, and the assembler is free to clobber \$X within area-1. This is avoided by using ".set_wr".

2.8.11 GPR A/B Bank Conflicts

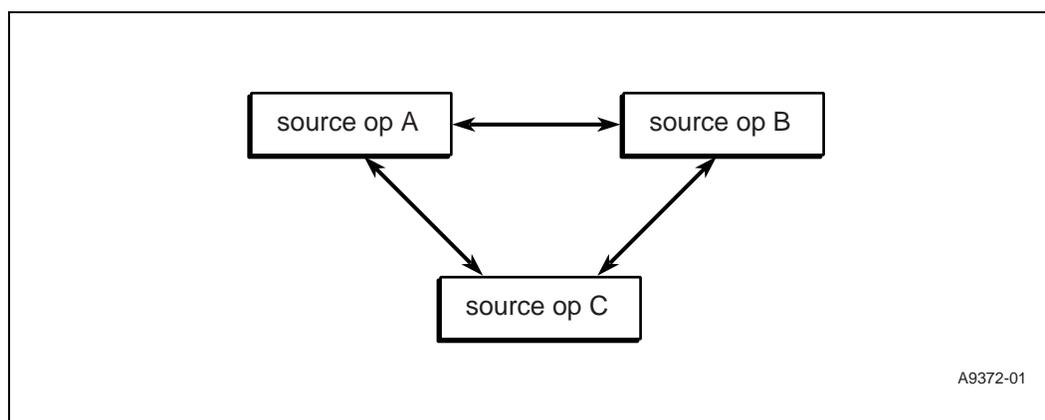
The GPRs are physically split into two banks called GPR_A and GPR_B. For each instruction, one GPR_A and one GPR_B operand can be read to form the two input operands to the execution datapath. The micro assembler allocates all GPRs into either the A or B GPR bank based on the specified pairings of GPR source operands as defined by the instruction source listing. Certain topological pairings of GPR source operands, cause an unresolved allocation problem. For example, consider the three instructions below:

```
alu[dest_op,source_op_a,+,source_op_b]
alu[dest_op,source_op_b,+,source_op_c]
alu[dest_op,source_op_a,+,source_op_c]
```

The first instruction above specifies that source_op_a and source_op_b must be on opposite GPR banks since these two operands must be simultaneously accessed. The second instruction specifies that source_op_b and source_op_c must also be on opposite banks for the same reason. Taken together, these two allocation constraints imply that source_op_a and source_op_c must be allocated to the same bank since they both require access in a bank opposite to that of source_op_b. However, the third instruction is in direct conflict with this since it requires source_op_a and source_op_c to be allocated on opposite banks.

These allocation problems can be understood and solved by some simple graphical analysis. The above example can be diagrammed as shown in [Figure 2-3](#).

Figure 2-3. Bank Allocation Diagram



Whenever a loop is formed with an odd number of sides, dual bank allocation is impossible. The solution is to rewrite the register relationships to either break the loop or form a loop with an even number of sides. In other words, rewrite the code to use a different GPR (causing the loop to break) or introduce another GPR in such a way as to add another side to the loop.

When the microassembler identifies this problem, it attempts to automatically solve the problem as described in [Section 2.8.11.1, “Automatic A/B Bank Conflict Resolution”](#) if, and only if, you have enabled this feature. By default, the feature is turned off. If the microassembler cannot solve the problem, it dumps the relevant topological information so that you can determine the best way to solve the problem. The design of the data path is such that the bank allocation problem is not shared by the transfer register banks because these registers can be accessed from either input port to the ALU.

2.8.11.1 Automatic A/B Bank Conflict Resolution

Automatically fixing A/B Bank conflicts is in some ways pretty straightforward. Where such a conflict exists, the assembler inserts an instruction to copy one of the registers to a temporary register and replaces the register reference with a reference to the temporary register.

The assembler attempts to minimize the number of added instructions, but it is likely that the programmer can do a better job of inserting such copies, since the programmer knows more about what the critical paths are within the program.

For implementation reasons, the assembler considers certain instructions as unmodifiable, and hence the registers in those instructions are unable to be changed to resolve an A/B bank conflict. The programmer can use the `#pragma optimize` mechanism (see [Section 2.11.2](#)) to further mark portions of the code as unmodifiable. This means that even if the option to automatically fix these conflicts is enabled, the assembler may not be able to fix certain conflicts.

By default this option is disabled. This is so that the user has to make an explicit request before the assembler starts inserting instructions into the user's program.

2.8.12 GPR Spilling

The basic approach for spilling GPRs to local memory is to map all of the virtual registers into physical registers. If the number of physical registers needed exceeds the available number, then spilling is required. Then, some number of the physical registers are selected to be located in local memory rather than in an actual GPR.

The steps that need to be taken are:

1. Determine which "physical registers" are able to be located in local memory.
2. Select a sufficient number of these that the remaining ones can fit into the actual physical registers.
3. Assign the spilled registers to local memory addresses.
4. Modify the source code to use local memory

If relative GPRs are spilled, then a non-spillable relative GPR is used to store the address of a local memory block to be used by that context. Absolute GPRs are not spilled.

Due to certain implementation restrictions, the assembler may not be able to use spilling to always resolve a "too many GPR" condition. Also, the programmer can use the `#pragma optimize` mechanism (see [Section 2.11.2](#)) to further mark portions of the code as unmodifiable.

While the assembler attempts to be efficient in modifying the code, it does not understand the program nearly as well as the programmer. While having the assembler handle spilling is convenient, it is likely that the programmer could use a similar approach to resolve the issue with less execution overhead.

By default this option is disabled. This is so that the user has to make an explicit request before the assembler starts inserting instructions into the user's program.

2.8.13 Lifetime Out-Of-Register Errors

A long present problem with the assembler has been that if the source code requires too many registers, it has been hard to figure out where the problem is occurring, and hence where changes should be made in order to reduce the number of required registers.

To help address this issue, there is now a command-line flag “-lr” which dumps the register lifetime information into a new file with a “.lri” extension. The reason that a new file is being used rather than using the .uci file is that if register allocation fails, then a .uci file is not produced.

Eventually, support for reading and displaying the .lri file will be integrated into the Workbench. However, until this is done and for non-Win32 systems, the user will have to manually examine the .lri file.

The .lri file basically consists of the raw source (post preprocessor) along with embedded “directives” giving the registers live at each line.

The directives are:

```
.%live_regs cnt addr gpr sr_xfer sw_xfer dr_xfer dw_xfer sig
```

This gives the count of live registers of the different types for the associated uword. The *addr* field gives the corresponding uword address (if the source file assembles successfully and the optimizer is not used). The next six numbers then give the counts in terms of relative registers. The count of GPRs can be non-integral due to absolute GPRs (since an absolute GPR occupies the space of either 1/8 or 1/4 that of a relative register, depending on how many contexts there are). “*sr_xfer*” refers to SRAM-read transfer registers, “*dw_xfer*” refers to DRAM-write-transfer registers, etc.

```
;%live_regs cnt addr gpr $R $W $$R $$W sig
```

This is a “comment” line that indicates what the different numbers from the “.%live_regs cnt” directive mean. It is particularly useful (as described later) when the counts are imported into a spreadsheet.

```
.%live_regs type addr name1 name2 name3 ...
```

This gives the actual list of live registers. The *type* field contains the same labels as from the “;%live_regs cnt” line; i.e. “gpr”, “\$R”, “\$W”, “\$\$R”, “\$\$W”, and “sig”. There is also a “@gpr” type, whose line lists all of the absolute GPR registers. Since absolute registers are essentially always live, rather than repeating this list throughout the file, the absolute GPRs are just listed once, at the beginning.

The *addr* field is as described above for “;%live_regs cnt”. The *names* are the names of the live registers. If the list is empty for a particular type, then that line is suppressed.

Figure 2-4. Example of a .lvr File

```

;-----
This is a separator between lines to make the file easier to
read.
An example output might be:
;%live_regs cnt addr gpr $R $W $$R $$W sig
.%live_regs @gpr 0 @g1
; lri_regs.uc: test of lri live range values
.reg tmp ; module
.reg global g
;-----
.%live_regs cnt 0 0.125 2 1 0 0 0
.%live_regs $R 0 $z $i2
.%live_regs $W 0 $z
nop
;-----
.%live_regs cnt 1 0.125 2 1 0 0 0
.%live_regs $R 1 $z $i2
.%live_regs $W 1 $z
immed[tmp, 0]

```

Here are some miscellaneous comments about the listing:

- The number of registers listed in the “.%live_regs xxx” lines may not match the count in the “.%live_regs cnt” lines for several reasons: for GPRs, the count also includes absolute GPRs, listed once at the beginning. Also, doubled signals increment the count by 2, but are only listed once.
- Due to implementation reasons, there may be a block of text with a separator and “.%live_regs cnt”, but which does not contain a real uword. For example:

Figure 2-5. Example of a .lvr File Without a Real uword

```

;-----
.%live_regs cnt 2 0 0 0 0 0 0
.if (x == 0)
;-----
.%live_regs cnt 2 1 0 0 0 0 0
.%live_regs gpr 2 x
alu[--,--,B,x]

```

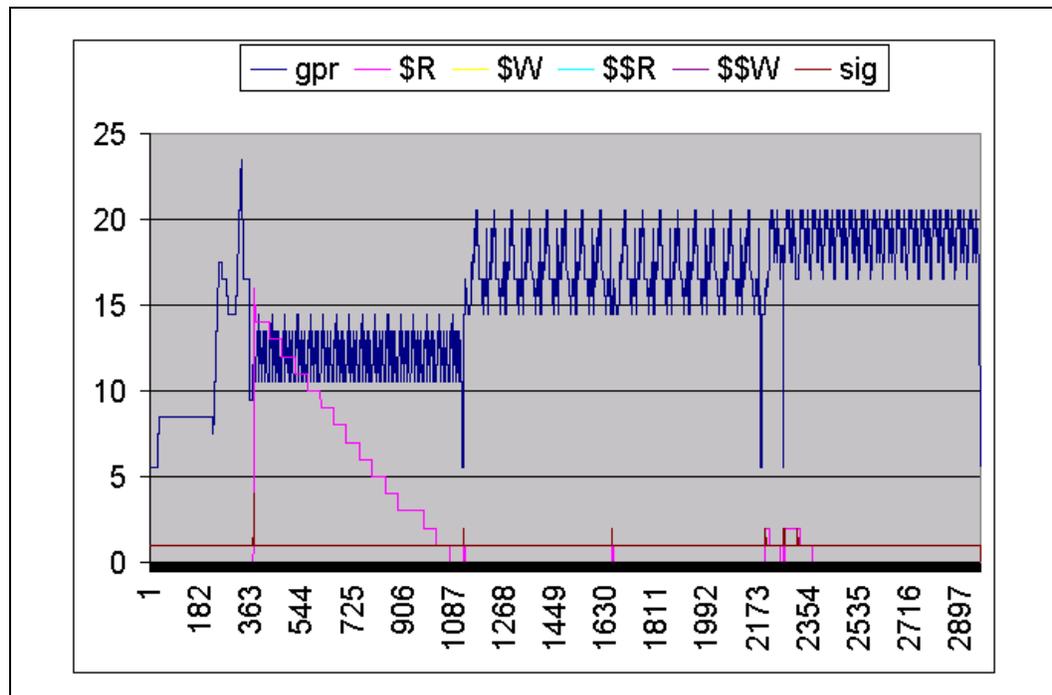
- Due to scoping, the same register may be repeated in the list. For example, there could be several registers named “tmp” defined in different nested scopes that are all live at the same time.
- The counts should be taken as indications of where the problem may lie and not as a hard-and-fast indicator of allocation success. For example, at the worst-case, there may be only 31 GPRs live, but it may still fail allocation due to A/B bank issues.

One of the most useful things to do with the .lri file is to filter it based on “.%live_regs cnt”, load it into a spread sheet, and then plot the counts. For example:

```
grep "%live_regs cnt" <file.lri >file.dat
```

One might then read that file into a spreadsheet, graph the data, and get a plot looking like Figure 2-6.

Figure 2-6. Lifetime Register Spreadsheet



From this, one can see that the GPR usage peaks around line 327 with a value of 23.5. Since this project was running in 4-ctx mode, there are 64 GPRs available, so about 40 of them should be available. If, however, the peak at line 327 exceeded 64, then register allocation would probably fail. In that case, by looking at what registers were live in that region, the programmer could hopefully identify some that could be made not live. Possible ideas for coping with excess register pressure would include:

- Rewriting the code to use fewer registers. For example, re-computing a value rather than using a value computed earlier. Similarly, delaying a calculation until just before a result is needed may help.
- Using absolute GPRs rather than relative ones where the value is not needed beyond a context-*arb*, or where the value is the same for all contexts. Note that absolute GPRs are essentially always live¹, so that an absolute virtual GPR corresponds to an actual physical register. Thus, absolute GPRs should be used for data with some permanence rather than for short-lived temporaries.

1. Absolute registers are typically used to communicate between threads. As such, most absolute registers are used by multiple threads. This means that most absolute registers need to be considered “used” whenever there is a context *arb*. This would make them live most of the time, so as a simplifying assumption, the assembler makes them live always.

- If the neighbor registers are not being used as a FIFO, they can become the repository of constants or pseudo-constants (values computed during startup but which don't change during the main loop).
- Using local memory to replace GPRs. This is particularly effective if one of the local memory index registers is available.
- It is also possible that the live-ranges of some registers are not being computed correctly due to incorrect use or lack of use of some of the directives. This would be indicated if a register was listed as live when it really was not. An example of this would be if a register was used, and then in a later section it was conditionally set and then later conditionally used (i.e. “correlated conditionals”). Without an appropriate “.set” directive, the register would be considered live from the first set to the last use, when in reality, it wasn't “live” between the two sets of references. In this case, the code produced would not be incorrect, but it may use more registers than necessary. Putting in the appropriate “.set” might reduce the register pressure.
- Another possibility is that there is an outright bug in the source code, and that by fixing the bug, the number of necessary registers would go down.

2.8.13.1 Transfer Register Lifetimes

It can be tricky for the assembler to compute when the lifetime ends for a write transfer register. The reason for this is that the instruction that ends the lifetime may only indirectly be related to the instructions using the register.

The same logic is used to extend the lifetime of read transfer registers, although typically this is not necessary as the registers are used after the I/O operation completes. It might be necessary if an I/O operation reads three words and only used the first and third. Then we would have to extend the lifetime of the second one.

The rule for transfer register lifetimes is that the lifetime extends beyond the I/O instruction until one of the following:

- An I/O instruction to the same queue generates a signal, and then that signal is consumed.
- The lifetime is explicitly ended via the `.io_completed` directive (see [Section 2.8.10](#)).
- The end of the defining block is reached.

Based on the I/O instruction, the queue may or may not be known. For MEv2, the queue is almost always unknown, with a few exceptions detailed below.

Note also that the word “queue” in this context refers to a logical queue, which may or may not represent a physical queue. For example, two operations that are stored on the same physical queue but which are processed in such a way that they may become reordered may be represented for the purpose of this section as being on two different queues.

2.9 Assembler Optimizer

The optimizer reduces execution time by performing the following optimizations:

1. Branch target optimization: Branches to unconditional branches are modified to directly branch to the ultimate branch target. This optimization is particularly useful for nested conditional statements. In addition, branches to consecutive addresses are removed. This optimization is useful when a conditional statement evaluates to false at assembly time.
2. Defer shadow filling: The defer shadows for branches and context swapping instructions are filled by moving instructions down into their defer shadows.
3. NOP removal: Unnecessary NOPs (NOPs that are not required to avoid violating timing constraints) are removed. For example, NOPs can be used to hide the latency between a local_csr_wr to an index register and the use of that index register. If the use of the index register were to be moved in a previous optimization step, then one or more of the NOPs might become unnecessary and would be removed.
4. NOP replacement: NOPs that cannot be safely removed in the above optimization step will be replaced by moving instructions down.

The optimizer is enabled via a Workbench dialog box or command-line option. Once enabled, the optimizer can be disabled and re-enabled for specific code sections by using the #pragma optimize directive described in section [Section 2.11.2](#).

The following nonsensical example illustrates each type of optimization.

Un-optimized code:

```

label#:
.if (x==y)
    .if (x==1)
        immed[z,0]
    .endif
.elif (x>y)
    immed[z,1]
.endif

#define i 1
#define j 2

.if (i==j)
alu[z,z,+,1]
.endif

alu[z,z,+,2]
local_csr_wr[ACTIVE_LM_ADDR_0,0]
nop
nop
nop
alu[z,z,+,*1$index0]

crc_be[crc_32,z,z]
nop
crc_be[crc_32,z,z]

```

```
br [label#]
```

Optimized code:

```
.if (x==y)
label#:
alu[--,10000!x,-,10000!y]
bne[1000_01#]
    .if (x==1)
        alu[--,10000!x,-,1]
    ; BRANCH TARGET OPTIMIZATION:  changed branch label from 1001_01# to 1000_end#.
        bne[1000_end#]
    .endif
.elif (x>y)
    1001_01#:
    1001_end#:
br[1000_end#], defer[1]

; BRANCH LATENCY FILL OPTIMIZATION:  the microword below was "pushed" down
; position
    immed[10000!z,0]
1000_01#:
alu[--,10000!x,-,10000!y]
ble[1000_02#]
    immed[10000!z,1]
.endif
.if (i==j)
; BRANCH TARGET OPTIMIZATION:  removed branch to next address.
; br[1002_01#]
; The following microwords are unreachable and have been commented out
; alu[10000!z,10000!z,+,1]
; End commenting out unreachable code
.endif
1000_02#:
1000_end#:
1002_01#:
1002_end#:
local_csr_wr[active_lm_addr_0,0]
nop
nop
; NOP REPLACEMENT OPTIMIZATION:  the microword below was "pushed" down 4
;positions
alu[10000!z,10000!z,+,2]
alu[10000!z,10000!z,+,*1$index0]
crc_be[crc_32,10000!z,10000!z]
; NOP OPTIMIZATION:  removed unnecessary NOP.
; nop
br[label#], defer[1]
; BRANCH LATENCY FILL OPTIMIZATION:  the microword below was "pushed" down 1
; position
crc_be[crc_32,10000!z,10000!z]
```

2.10 Assembler Directives

This section defines the Assembler directives are supported by the assembler

2.10.1 Summary of Directives

Table 2-11. Assembler Directives (Sheet 1 of 3)

	Type	Directive	Arguments Expanded	Description
preprocessor	Assembler Loops	#for	No	Repeat following lines based on a const expression.
		#repeat, #while	Yes	Repeat following lines based on a const expression.
		#endloop	N/A	End repeated lines.
	Assembler Macros	#macro	No	Start defining a macro.
		#endm	N/A	Finish defining a macro.
	Conditional Assembly	#ifdef, #ifndef	No	Conditionally skip following lines.
		#if, #elif	Yes, including "defined(name)"	Conditionally skip following lines based on a const expression.
		#else, #endif	N/A	Conditionally skip following lines.
	Error Reporting	#error		Displays a message and optionally aborts processing.
	File Inclusion	#include	No	Start reading lines from another file.
	Token Replacement #define	#define	No	Define an expandable token.
		#define_eval	Yes	Define an expandable token to a const expression.
		#undef	No	Undefine an expandable token.
	Structured Assembly	.if, .elif	Yes	Generate ME instructions that are executed at runtime.
		.if_unsigned, .elif_unsigned	Yes	
		.else, .endif	N/A	
		.while	Yes	
		.while_unsigned	Yes	
		.endw	N/A	
.repeat		N/A		
.until		Yes		
.until_unsigned		Yes		
.break, .continue		Yes		

Table 2-11. Assembler Directives (Sheet 2 of 3)

	Type	Directive	Arguments Expanded	Description
Assembler	Import Variable	.import_var		Defines a set of symbolic names that will be imported by the linker.
	Local Regions	.begin		Defines a region of code and a new set of registers that are only visible within that region.
		.end		
	Local Regions old form	.local		Older form of region definition, supported for compatibility with older code.
		.endlocal		
	Register declaration	.reg		Declares register symbolic names which will be allocated to physical registers by the assembler.
	Initialization	.init		Initializes all or part of a block of memory or a register.
	Manage register and signal usage warnings	.set .set_sig .set_rd .set_wr		These directives generate no code, affecting "used before set" warnings for registers and signals.
	Extend or define register lifetime	.use .use_rd .use_wr		These directives generate no code, but tell the assembler registers in question are being used up to the point where they are not.
	Signal completion of I/O operation	.io_completed .io_completed_type		These directives generate no code, telling the assembler that an I/O operation has been completed.
	Manual Register & Signal Specification	.addr		Preferred form to manually allocate registers and signals.
	Old form of Manual Register Specification	.areg .breg .\$reg .\$\$reg		Older forms to manually allocate registers.
	Optimization Directives	#pragma optimize		This directive provides precise control over the optimizations performed by the assembler.
	Subroutine Definition	.subroutine .endsub		Define a region of code containing a subroutine.
	Memory Allocation	.local_mem .global_mem		Allocate a block of memory for a shared resource (i.e., SRAM, DRAM, SCRATCH, or local memory).
	Local Memory Mode	.local_mem0_mode .local_mem1_mode		Set local memory mode to either abs or rel
Number of Contexts	.num_contexts		Set context mode for the microengine to 4 or 8 threads.	
Neighbor Mode Directive	.init_nn_mode		Set the initial NN_MODE in the CTX_ENABLE register, either "neighbor" or "self". If two different values are specified, an error results. If no value is specified, the default value is "neighbor". Note that this is handled by the loader and does not generate any microcode.	
Transfer Order	.xfer_order .xfer_order_rd .xfer_order_wr		Define an ordering of transfer registers. .xfer_order_rd and .xfer_order_wr are obsolete -- do not use.	

Table 2-11. Assembler Directives (Sheet 3 of 3)

	Type	Directive	Arguments Expanded	Description
Linker	Linker Directives	.image_name .entry .page .ucode_size		The following directives are passed through the assembler to the linker and are listed without comment.

2.11 Directives Definitions

2.11.1 Token Replacement (#define, #undef)

The #define directive causes subsequent instances of the identifier to be replaced with the token string. After this directive, the identifier is referred to as an expandable token. The #undef directive removes the definition for the given identifier.

Macros and expansion tokens share the same name space (i.e. it is illegal to have a macro with the same name as a #define token).

Instruction Format

```
#define identifier token-string
#define identifier(arg1, ...) token-string
#undef identifier
```

Note that there cannot be any spaces between “identifier” and the “(“.

This facility is essentially equivalent to that of the C-processor. The expansion of arguments, however, is handled in a similar manner to arguments for macros defined with #macro.

There is a variation of #define that defines the replacement for the identifier as the value of const-expr. If the identifier is previously defined, then it is redefined. This is primarily designed for use within assembly loops. The constant expression can evaluate to either a numeric constant or to an identifier. For details on identifier expressions, see [Section 2.6.3, “Constant Expressions \(const-expr\)”](#).

Instruction Format

```
#define_eval identifier const-expr
```

Examples

Note that the following would be valid:

```
#define_eval foo 123
#define_eval foo foo + 123 /* assuming foo is initially numeric */
#define_eval foo abc
#define_eval foo strleft(abcd, 2) /* evaluates to ab */
```

but the following would not be:

```
#define_eval foo abc def
```

This is because "abc def" is not a valid expression.

2.11.2 Optimization Directives

The "#pragma optimize" directive allows the various assembler optimizations to be individually disabled or re-enabled for specified blocks of code. The syntax is as follows:

```
#pragma optimize( "optimization-list", {on | off} )
```

The optimization-list specifies the types of optimizations to enable or restore. An empty list specifies all types of optimizations. Values are:

Table 2-12. Optimization List for #pragma optimize Directive

Optimization Type	Description
b	Fill defer shadows (bubbles) (see Section 2.9 , item 2).
n	Optimize NOPs (see Section 2.9s. , items 3 and 4).
t	Optimize branch targets, (see Section 2.9 , item 1).
d	All of the above execution speed optimizations (b,n, and t), optimize branch targets, fill defer shadows, and replace/remove NOPs (see Section 2.9). Equivalent to specifying "tbn".
f	Try to automatically fix A/B Bank conflicts (see Section 2.8.11).
s	Try to automatically spill GPRs into local memory (see Section 2.8.12).
empty string	Equivalent to specifying all of the above.

Examples

```
#pragma optimize("fs", off) ;Turn off A/B conflict fixing and spill
                             ;optimizations
#pragma optimize("", on) ;Turn on all optimizations
```

Specifying "off" marks the instructions following the #pragma optimize directive as not subject to optimization. Specifying "on" marks subsequent instructions as subject to the specified optimizations.

These directives can nest. That is, conceptually, there is a count for each type of optimization that starts at zero. When a "#pragma optimize("optimization-list", off)" directive is seen, the count is incremented. When an "#pragma optimize("optimization-list", on)" directive is seen, the count is decremented. Optimizations are disabled whenever the count is greater than zero. The rationale for this is that a macro may want to disable optimizations for a sequence within itself, but if the caller had previously disabled optimizations, then the macro should not "accidentally" turn them back on. It is an error if the count ever goes negative. For example, it would be an error if the first such directive seen in a function is "#pragma optimize("optimization-list", on)". Note that this directive only turns optimizations off or restores them; it cannot turn them on if the user did not specify optimizations via the command line arguments.

2.11.3 Loops

2.11.3.1 For Loops (#for, #endloop)

Repeats the text-lines with the identifier taking on each of the listed values. The identifier behaves as if it were the target of a #define. That is, it is set at a global scope, overwriting any existing definition. After the loop is ended, the identifier continues to exist with the last value specified in the directive.

Instruction Format

```
#for identifier [arg1, arg2, ...]
text-lines
#endloop
```

Examples

```
#for id[1,2,3]
immed[reg, id]
#endloop
```

Assembles to:

```
immed[reg, 1]
immed[reg, 2]
immed[reg, 3]
```

2.11.3.2 Repeat Loops (#repeat, #endloop)

Repeats text-lines for the indicated number of times. If const-expr evaluates to a negative number, an error results.

Instruction Format

```
#repeat const-expr
...text-lines...
#endloop
```

Examples

```
#repeat (2)
immed[reg,5]
#endloop

Assembles to:
immed[reg,5]
immed[reg,5]
```

2.11.3.3 While Loops (#while, #endloop)

This repeats text-lines as long as the expression evaluates to a nonzero value.

Instruction Format

```
#while const-expr
...text-lines...
#endloop
```

Examples

```
#define LOOP 3
#while (LOOP > 0)
immed[reg1,5]
#define_eval LOOP (LOOP-1)
#endloop

Assembles to:
immed[reg,5]
immed[reg,5]
immed[reg,5]
```

2.11.4 Macros (#macro, #endm)

A macro is a series of directives and instructions grouped together as a single command. Macros are identified by an identifier and optional parameters can be passed to the macro for processing.

Instruction Format

```
#macro identifier(arg1, arg2, ...)
...text-lines...
#endm
```

References to macros are defined by:

Instruction Format

```
[label] identifier[arg1, arg2, ...]
```

These references are replaced by text-lines, and instances of the parameters in the text-lines are replaced by the macro arguments. Any leading or trailing spaces in macro arguments are ignored by the assembler. If a label is supplied before the macro reference, it is inserted before the first line in text-lines. Additionally, labels appearing in text-lines have the label from the reference prepended. If there is no label in the reference, the labels appearing in text-lines have an automatically generated prefix prepended.

For example, consider the code fragment:

Examples

```
#macro test1[param1,param2]
immed[param1,param2]
mylab#:
alu[reg,reg1,+,param1]
br=0[mylab#]
#endm
l1#: test1[reg2, 5]
l2#: test1[reg2, 6]
```

These two references would expand into:

Examples

```
l1#:
    immed[reg2, 5]
l1_mylab#:
    alu[reg,reg1,+,reg2]
    br=0[l1_mylab#]
l2#: immed[reg2, 6]
l2_mylab#:
    alu[reg,reg1,+,reg2]
    br=0[l2_mylab#]
```

Note that a comma enclosed within a set of parenthesis or brackets within the token list in the macro reference does not serve to separate the items—it is considered part of the item itself. For more information about commas within parentheses and brackets, see [Section 2.6.5](#)

Optionally, a signal can be replaced with the construct

```
signals(sig1, sig2, ...)
```

Where *sigN* can also be of the form `signals(...)`.

The effect is the same as if *sigN* were entered as an operand of the `ctx_arb`. For example:

```
ctx_arb[siga, sigb, signals(sigc, sigd, signals(sige, sigf)), sigg, signals]
```

would be equivalent to:

```
ctx_arb[siga, sigb, sigc, sigd, sigf, sigg]
```

Note that “signals” with no arguments becomes effectively a null signal, although this can also be done using “sig_none”. This mechanism is primarily designed to pass a variable number of signals into a macro using a single argument.

Examples

```
#macro foo[sigs]
ctx_arb[sigs]
#endm
...
foo[signals(sig1,sig4)]
```

Examples

```
#macro add(out_dst, in_src_a, in_src_b)
inside_label#:
alu[out_dst, in_src_a, +, in_src_b]
#endm

The following code:
label_outside#:
immed[reg,4]
add(result, reg, 6)

Assembles to:
label_outside#:
immed[reg,4]
add(result, reg, 6)
m000_inside_label#:
alu_shf[result, reg, +, 6,0]
```

2.11.5 Conditional Assembly (#ifdef, #if, #else, #elif, #endif)

Conditional directives are similar to the directives used with a standard CPP. If the identifier is defined (for #ifdef), then the following text-lines are included in the output. If the identifier is not defined, the else lines are included. There may be multiple #elif clauses. If none of the #if or #elif clauses are true and there is an #else clause, those text lines will be included. Identifier replacement is done within the constant expression.

Conditional assembly constructs can nest.

Table 2-13. Condition directives

#if	#else	#elif
#ifdef	#ifndef	#endif

Instruction Format

```
#ifdef identifier ; ifdef can be replaced with ifndef
...text-lines...
#else
...text-lines...
#endif
```

Instruction Format

```
#if const-expr
...text-lines...
#elif const-expr
...text-lines...
#else
...text-lines...
#endif
```

Examples

```
#define test_val 5

#if (test_val > 3)
immed[reg,1]
#elif (test_val < 3)
immed[reg,2]
#else
immed[reg,3]
#endif

Assembles to
immed[reg,1]

Changing the test_val to 0 assembles to:
immed[reg,2]

Changing the test_val to 3 assembles to:
immed[reg,3]
```

2.11.6 Error Reporting (#error)

Print the specified string with the specified error severity level

Instruction Format

```
#error [severity] "message string"
```

The severity is an optional integer with a default value of 4 and the following meanings:

Table 2-14. Error Reporting Severity Levels

severity level	meaning
0	Information
1	Warning
2	Error
3	Error; abort processing this file
4	Error; abort all processing

2.11.7 File Inclusion (#include)

The line of code with the include directive will be replaced with the contents of the named file. There is no default extension. The correct extension must be supplied. The search path consists of the current directory of the file containing the directive, directories specified on the command line, and directories specified via environment variables. Files that are included can themselves include other files. Note that there is no form using angle brackets (<>), which is different from CPP.

Instruction Format

```
#include "filename.ext"
```

2.11.8 Import Variable (.import_var)

Defines a set of symbolic names that will be imported by the linker. These directives may be used wherever a numeric constant may be used. A typical example would be assigning the value of a symbol to a register via the immed opcode.

The result of this directive is that the portions of the assembled instructions that would have contained the constant value are left void, and directives are passed to the linker so that the linker/loader can insert the constants in the appropriate places. A potential problem is that the constants being imported may be as large as 32 bits, whereas immediate data within the instruction are limited to smaller values. Thus, the programmer may want to load some part of the imported constant other than the lower order bits. To allow this, the reference to the imported symbol may be suffixed with >>num, which indicates that the imported value should be right-shifted by the specified amount (num), and the low order bits of the result are used. If no such suffix is applied, the effect is the same as a suffix of >>0.

Imported variables can also be used in constant expressions, for example:

```
immed[reg, ((sym1 >> 8) & 0xFFFF), <<8]
```

Instruction Format

```
.import_var sym1 sym2 ...
```

The scope of imported variables is essentially global, although the symbol can only be referenced after its definition.

Import variable names should be prefixed with “i\$” so that the preprocessor function `isimport()` (see [Table 2-4](#)) can correctly identify the name as an import variable. If `isnum()` is used on an import variable which does not have the “i\$” prefix, the `import_var` directive will generate a warning (see [Appendix A, “UCA Warnings”](#) warning [A.53](#)).

2.11.9 Code block directive (.begin, .end)

The block directives define a region of code and a new set of registers that are only visible within that region.

Blocks are explicitly delimited by “.begin” and “.end” directives:

Instruction Format

```
.begin  
.reg name1 name2 ...  
...  
.end
```

An older syntax for defining block structure is still supported, but should not be used for new code:

Instruction Format- older supported form

```
.local reg1 reg2...  
.endlocal
```

2.11.10 Manual Register Allocation (.addr)

In rare cases, a programmer may want to manually allocate registers. As shown below, specifying a relative register allocates an entire row of addresses, while specifying an absolute register allocates only one cell.

Table 2-15. Register Mapping - Context Relative to Absolute

Context Relative address	Absolute Address							
	Context Relative 0-15 for ctx:							
	0	1	2	3	4	5	6	7
0	0	16	32	48	64	80	96	112
1	1	17	33	49	65	81	97	113
2	2	18	34	50	66	82	98	114
3	3	19	35	51	67	83	99	115
4	4	20	36	52	68	84	100	116
5	5	21	37	53	69	85	101	117
6	6	22	38	54	70	86	102	118
7	7	23	39	55	71	87	103	119
8	8	24	40	56	72	88	104	120
9	9	25	41	57	73	89	105	121
10	10	26	42	58	74	90	106	122
11	11	27	43	59	75	91	107	123
12	12	28	44	60	76	92	108	124
13	13	29	45	61	77	93	109	125
14	14	30	46	62	78	94	110	126
15	15	31	47	63	79	95	111	127

Caution should be used to avoid allocating the same register location more than once. For example, it is probably wrong to assign a relative GPR to 2, and an absolute GPR to 50, because relative GPR 2 allocates the entire row of absolute GPR addresses: 2, 18, 34, and 50.

Note: If any GPR is manually allocated, all GPRs must be manually allocated. The same is true for neighbor registers. However, transfer registers and signals can be partially manually allocated and the rest automatically allocated.

Manual register and signal allocation is now done with the following directive:

```
.addr name address [bank]
name:      Name of register or signal
address:   Address of register
bank:     In the case of GPRs, bank should either be "a" or "b"
```

The type of register determines where the register is allocated. If *name* matches both a GPR and a signal, then if the bank token is present, it is assumed to be the GPR; otherwise it is assumed to be the signal.

Note that this directive does not declare a register or signal. If register declaration is required, then the registers and signals must be declared before being allocated.

A manually allocated register/signal should probably be considered volatile (see [Section 2.8.2.1, “Preferred Register Declaration Syntax”](#) and [Section 2.8.6, “Signal Declarations”](#)), but there may be cases (e.g. for debugging) where the volatile declaration is not needed.

Assigning two registers to the same address will result in a warning. The reason it is not considered an error is that there may be cases where it is useful.

For compatibility with earlier releases, the existing style of register declarations will still be supported. For example:

```
.areg name 4
```

is equivalent to:

```
.addr name 4 a
```

The instruction formats of the older style of register declarations are shown below.

Instruction Format

```
.areg reg num
.breg reg num
.$reg reg num
.$$reg reg num
```

2.11.11 Memory Allocation Directives

These directives allow you to “allocate” a block of memory from a shared resource (e.g. SRAM, DRAM, SCRATCH). The first two directives allocate the block:

```
.local_mem [keyword] name region size [align]
.global_mem name region size [align]
```

keyword: Optionally, the keyword **global** may be used.
name: Name of block being defined
region: In which region is the block being allocated. This can take one of the values: **SRAMn**, **DRAM**, **SCRATCH**, or **LM**, where *n* is 0, 1, 2, ... and specifies the SRAM channel.
size: Size of region in bytes
align Required alignment (in bytes). Default is 4.

The difference between these two directives is in what happens if two or more microengines declare a block with the same name. Any which are declared with `.local_mem` result in an individual block for the ME. All which are declared with `.global_mem` refer to the same block.

The result within the assembler is that there is a symbolic constant, defined by *name*, that behaves as if it were an imported symbol and whose value is the base address of the block.

The region LM corresponds to the microengine's local-memory and is only valid for `.local_mem` (it can't be shared between microengines). For this region, the alignment must be a multiple of 4.

If two blocks are declared in the same scope, then the parameters must be "compatible", that is all parameters must either be equal or zero. It is an error to have all of the parameters zero. For example, a programmer could define the same block several times with sizes of 10, 10, 0, 0, 10, and 0. But they could not define the same block with sizes of 10 and 14.

The scoping of *name* is the same as for the `.reg` directive, as modified by the **global** keyword (see [Section 2.8.2, "Register Declarations"](#))¹. To reference a block defined in another module, you should define a global block with the same name. In this case, either the sizes must match, or all but one of the sizes must be zero.

Note that imported symbols and memory block names share the same name-space as GPR register names. Imported symbols (and hence memory block names) take precedence over register names. That is, if there is (in scope) a GPR with the name XXX and a memory block with the name XXX, then references to XXX will be taken as the memory block.

2.11.12 Memory Block and Register Initialization

This directive is used to initialize part or all of a memory block or a register:

```
.init name[+offset] value1 value2 ...

name:      Name of block or register being referenced
offset:    A byte offset from the start of the block. The offset must be a
              multiple of 4.
value:    Value to be stored at the designated address.
```

The offset can only be used if name references a memory block. If name references a register, then the register must be declared with a global or module scope, and only one value can be specified.

Each value defines one 32-bit word. Each value can take one of four forms:

- A numeric constant
- A previously defined memory block name (either in this region or another)
- A previously defined memory block name plus a constant (e.g. "x+5").
- A constant expression contained within parenthesis.

Note that if the register is a transfer register that is declared as both a READ and WRITE register, then both physical registers will be initialized. When initializing a relative register, the register is initialized for all contexts.

A register or memory block can be initialized multiple times as long as the same value is used. In some cases, such as when the initialization value involves remote values, the assembler cannot perform this check and the check will be deferred to the linker/loader.

The actual initialization is handled by the micro-engine loader at load-time and no code is generated by this directive.

1. More particularly, if ".local_mem" is used, then the block name is mangled to make different references unique. If ".global_mem" or ".local_mem global" is used, then the block name is not mangled.

2.11.13 Local Memory Mode Directives

```
.local_mem0_mode type
.local_mem1_mode type
type:      Either abs or rel
```

This sets the mode of the specified local memory index register. If the mode is not specified, it defaults to “rel” (relative).

If the mode is set to “abs” (absolute), then all contexts share one physical register when accessing local memory. When set to “rel”, each context has its own register. Note that the two registers (0 and 1) may be set differently.

This directive may be issued multiple times for a particular register. If any of the settings conflict, however, an error is generated.

The results of this directive are passed along to the loader so that the MEv2 CSRs may be set correctly.

2.11.14 Number of Contexts Directive

```
.num_contexts n

n: Number of contexts (8 or 4)
```

For MEv2, if this directive is not specified, it defaults to 8.

This directive can appear any number of times throughout the module, but if any conflict, the assembler will generate an error.

2.11.15 Initial Next Neighbor Mode Directive

```
.init_nn_mode mode

mode:  either "self" or "neighbor"
```

This sets the initial NN_MODE in the CTX_ENABLE register. If two different values are specified, an error results. If no value is specified, the default value is “neighbor”.

Note that this is handled by the loader and does not generate any microcode.

2.11.16 Operand Synonym (.operand_synonym)

Note: This directive is no longer supported, and use of it will generate an error. Use #define instead. The following description is included to assist people who wish to understand and convert old code which uses operand_synonym.

This directive defines synonym values for operands (for example, instruction token values specified inside square brackets ([])) may be specified. This allows multiple names to exist for a particular register, and allows numeric constants to be described in symbolic form.

In the syntax example, this directive defines a synonym for `new_syn_name` that is defined from this point in the file to the end. After this directive, references to `operand_name` are equivalent to references to `new_syn_name`. Note that references to `operand_name` above this directive refer to a distinct register that is different from `new_syn_name`.

Note: To share registers, use a local regions command rather than an operand synonym.

There is no reason to use this directive; it is mainly supported for compatibility with earlier versions. Use different registers rather than two names for the same register and use `#define` to define symbolic constants.

Instruction Format

```
.operand_synonym operand_name new_syn_name
.operand_synonym name value
```

2.11.17 Structured Assembly

The goal of these directives is to allow programmers to organize the control flow of their programs into structured blocks as opposed to a sea of goto statements. These directives begin with a period (`.`), not a pound sign (`#`).

Several variations are described in the following sections. These reference a conditional-expression, which is described in [Section 2.11.17.5](#).

2.11.17.1 Conditional (`.if`, `.elif`, `.else`, `.endif`, `if_unsigned`, `.elif_unsigned`)

The `if`, `elif`, `else` directives generate instructions specifying that if the conditional expression is true, then the `if` text-lines will be executed.

Instruction Format

```
if-part [elif-part]* [else-part] endif-line
if-part:if-line text-lines
if-line:.if cond-expr
elif-part:elif-line text-lines
elif-line:.elif cond-expr
else-part:else-line text-lines
else-line:.else
endif-line:.endif
```

To do an unsigned comparison, one would use these alternate directive forms:

```
.if_unsigned cond-expr
.elif_unsigned cond-expr
```

2.11.17.2 Repeat Loops (.repeat, .until)

The .repeat and .until directives generate instructions specifying that the text-lines will be executed until the conditional expression is true. The conditional expression is evaluated after the loop is executed.

Instruction Format

```
repeat-line text-lines until-line  
repeat-line:.repeat  
until-line:.until cond-expr
```

To perform an unsigned comparison, use the following directive in place of the .until
.until_unsigned cond-expr

2.11.17.3 While Loops (.while, .endw)

The .while and .endw directives generate instructions specifying that the text-lines will be executed as long as the conditional expression is true. The conditional expression is evaluated before the loop is executed.

Instruction Format

```
while-line text-lines endw-line  
while-line:.while cond-expr  
endw-line:.endw
```

To perform an unsigned comparison, use the following directives in place of the .while:
.while_unsigned cond-expr

2.11.17.4 Break and Continue

The break and continue directives generate instructions that skip the remaining portions of .while and .repeat loops. The .break directive causes the loop to terminate, and execution continues at the code following the loop. The .continue directive will cause the next iteration of the loop to occur, provided that the loop condition allows it. That is, for a while loop, the next iteration will begin if the loop condition is still true. For a repeat loop, the next iteration will begin if the loop condition is false.

Instruction Format

```
.break  
.continue
```

2.11.17.5 Conditional Expressions

A condition-expression is used to select the path for control flow in the previous constructs. The expression has four forms:

- A constant
- A comparison of a register with either another register or a constant
- A comparison of a function against a constant

- A testing of the condition codes.

In the function form, the function can either access one bit of a register, one byte of a register, the carryout, the context, an input state, or a signal. Conditional expressions have the following form:

```

Syntax:
cond-expr: cc-expr
           non-cc-expr
non-cc-expr: const
            eq-expr
            gt-expr
            log-expr
gt-expr: (reg-expr reg-op const)
         (const reg-op reg-expr)
         (reg-expr reg-op reg)
         (reg reg-op reg-expr)
reg-expr: reg
         reg shift-op const
         reg shift-op reg
shift-op: >>
         <<
reg-op:  ==
         !=
         >
         >=
         <
         <=
         &
eq-expr: (func func-op const)
         (const func-op func)
         func
func:    BIT(reg, pos)
         !BIT(reg, pos)
         BYTE(reg, pos)
         COUT()
         !COUT()
         CTX()
         INP_STATE(state)
         !INP_STATE(state)
         SIGNAL(signal)
         !SIGNAL(signal)
func-op: ==
         !=
cc-expr:  =0
         !=0
         0
         =0
         <0
         <=0
log-expr: (non-cc-expr log-op non-cc-expr)
log-op:   ||
         &&

```

This defines a condition-expression, used to select the path for control flow in the previous constructs. The expression has four forms: a constant, a comparison of a register with either another register or a constant, a comparison of a function against a constant, or a testing of the condition codes. In the function form, the function can either access one bit of a register, one byte of a register, the carryout, the context, an input state, or a signal.

When a function appears by itself, it is the same as function != 0. The not operator (!) can be applied to functions that return a boolean result. Note that the gt-expr forms result in an ALU opcode followed by a branch opcode. Note also that all numeric constants can be replaced by constant expressions.

The pos and const values must fall into the following ranges (Table 2-5):

Table 2-16. pos and const Values

Function	Pos	Const
bit	0—31	0—1
byte	0—4	0—255
cout	n/a	0—1
ctx	n/a	0—7
inp_state	n/a	0—1
signal	n/a	0—1
n/a = not applicable		

2.11.17.6 Errors

If the structured assembly constructs are not followed by valid instructions, you may get an error of the form:

```
ERROR: Specified label, "l000_01#", does not exist.
```

This is caused by an attempt to branch out to the following (non-existent) instruction. The solution is to provide at least one valid microword after these constructs.

For example, the following program fails with the above error:

```
.reg reg
.if (reg==1)
.endif
```

However, the error goes away if there is a nop added at the end:

```
.reg reg
.if (reg==1)
.endif
nop
```

The cause of the problem and the solution should be obvious if one examines the generated .ucp file.

2.11.18 Structured Assembly Usage Considerations

The while loop trades off microwords in favor of execution speed. In particular, the condition is instantiated at the start of the loop and then repeated at the end of the loop. This avoids an extra branch latency at the cost of one to three microwords. For example, the code segment:

```
.while (reg > 0)
alu[reg,reg,-,1]
.endw
```

Generates code of the form:

```
alu[--,reg,-,0]
br<=0[1000_end#]
1000_start#:
alu[reg,reg,-,1]
1000_cont#: alu[--,reg,-,0]
br>0[1000_start#]
1000_end#:
```

Notice that the alu[--, reg and br instructions are repeated twice.

This is not the case for the repeat/until directives. These directives generate smaller code, important for tight loops. For example, if one were to try to loop until a signal is set using a while loop:

```
.while (!signal(sram_sig))
.endw
```

Generates code of the form:

```
br_signal[sram_sig,1000_end#]
1000_start#:
1000_cont#:
br_!signal[sram_sig,1000_start#]
1000_end#:
```

On the other hand, if you coded it as a repeat/until loop:

```
.repeat
.until (signal(sram_sig))

You get the following much cleaner code:

1002_start#:
1002_cont#: br_!signal[sram_sig,1002_start#]
1002_end#:
```

2.11.19 Warning Directives

Each warning will be associated with a numeric ID and a *warning-level*. The level indicates the degree of seriousness of the warning. Level-1 is the most serious. Level-4 is the least serious. The default is level-3. Additionally, there is a warning level parameter for the program run, which determines which warnings are generated and which are suppressed. If the warning level is set to *n*, then all warnings whose level is $\leq n$ will be presented. Those $>n$ will be suppressed.

That is to say, warnings that may be of interest to the programmer under some circumstances, but which would just be “noise” most of the time would be level-4 warnings. So by default, these would not appear, but the user could cause these to be visible if desired. The warning level is set with the following command line options:

- w Same as -W0
- Wn Set the warning level to n with $0 \leq n \leq 4$
- WX Make all warnings (at or below the level given by -Wn) errors

Note that setting the warning level to zero effectively disables all warnings. Note also that -Wn and -WX can both appear together.

Warnings can be modified using the following directives:

```
#pragma warning(spec:list,...)
#pragma warning(push)
#pragma warning(push,n)
#pragma warning(pop)

spec:      Warning specifier from following table
list:      One or more space separated warning identifier numbers
```

Spec	Meaning
once	Only display the listed warnings once.
default	Reset the warning attributes to their default value.
1,2,3,4	Apply the indicated warning level to the listed warnings
disable	Disable the listed warnings
error	Treat the listed warnings as errors.

The push and pop pragmas save and restore the warning attributes and global warning level on a stack. The (push,*n*) pragma also sets the global warning level to *n*.

In the case that a warning applies to multiple lines, and these lines have different attributes, the most conservative attributes will be used. That is, if either line treats it as an error, the warning will be assigned to that line, otherwise the line which applies the lowest level to that warning will be used. In short, in such cases, the only way to suppress such a warning is to suppress it at both lines.

The approximate meanings of the warning levels are:

Level	Meaning
1	Severe Warning: The code is probably incorrect. This is not an error because assembly can continue and it is slightly possible that the code may not be incorrect.
2	Medium Warning: This is the default level for warnings.
3	Minor Warning: The code can probably assemble correctly, but it might contain bugs. The programmer should probably take a look at these to be safe, but these can probably be safely ignored.
4	Informative Warning: This is a low-level warning that may warn about possible performance issues or other minor things that do not directly affect the correctness of the program.

2.12 Subroutine Definition (.subroutine, .endsub)

Define a region of code containing a subroutine. These directives do not directly affect the generated code. They are used to correctly define the live range of registers as described in Section 2.8.5, “Register Lifetime Details”.

Instruction Format

```
.subroutine
.endsub
```

2.13 Linker Directives

The following directives are passed through the assembler to the linker and are listed without comment:

Table 2-17. Linker Directives

Directive	Description
.image_name .entry .page .ucode_size	The following directives are passed through the assembler to the linker and are listed without comment.



MEv2 Instruction Set

3

For quick reference, the microengine instruction set is summarized and grouped by function in [Table 3-1](#). In the sections that follow, the instructions are defined in alphabetical order.

Table 3-1. Summary of Microengine Instructions

Instruction	Description	Section
General Instructions		
ALU	Perform an ALU operation.	Section 3.2.1
ALU_SHF	Perform an ALU and shift operation.	Section 3.2.2
ASR	Perform an arithmetic right shift on a register	Section 3.2.3
BYTE_ALIGN_BE, BYTE_ALIGN_LE	Concatenate data in two registers and put any four bytes into the destination register.	Section 3.2.11
CRC_LE, CRC_BE	Compute CRC_LE (Little Endian) . Compute CRC_BE (Big Endian).	Section 3.2.21
DBL_SHF	Concatenate two 32-bit words, shift the result, and save a 32-bit word.	Section 3.2.23
MUL_STEP	Multiply two unsigned numbers.	Section 3.2.38
FFS	Determine position number of LSB set in a register.	Section 3.2.26
POP_COUNT	Determine the number of bits set in a register.	Section 3.2.41
IMMED	Load immediate 16-bit word and sign extend or zero fill with shift.	Section 3.2.29
IMMED_BO, IMMED_B1, IMMED_B2, IMMED_B3	Load immediate byte to a field.	Section 3.2.30
IMMED_WO, IMMED_W1	Load immediate 16-bit word to a field.	Section 3.2.31
LD_FIELD, LD_FIELD_W_CLR	Load byte(s) into specified field(s).	Section 3.2.33
LOAD_ADDR	Load instruction address.	Section 3.2.34
LOCAL_CSR_RD, LOCAL_CSR_WR	Read and write Microengine Local CSRs.	Section 3.2.35 , Section 3.2.36
NOP	No operation and virtual no operation	Section 3.2.39
Branch and Jump Instructions		
BCC	Branch on condition code.	Section 3.2.4
BR	Branch unconditionally.	Section 3.2.5
BR_BCLR, BR_BSET	Branch on bit clear or bit set.	Section 3.2.6
BR=BYTE, BR!=BYTE	Branch on byte equal or not equal to literal.	Section 3.2.7
BR=CTX, BR!=CTX	Branch on current context.	Section 3.2.8
BR_INP_STATE, BR_IINP_STATE	Branch on event state (e.g., SRAM done).	Section 3.2.9
BR_SIGNAL, BR_!SIGNAL	Branch if signal deasserted.	Section 3.2.10
JUMP	Jump to label.	Section 3.2.32
RTN	Return from a branch or a jump.	Section 3.2.42

Table 3-1. Summary of Microengine Instructions (Continued)

Instruction	Description	Section
I/O and Context Swap Instructions		
DRAM (Read and Write)	Move data between the DRAM and the Microengine D-Transfer registers.	Section 3.2.24
DRAM (RBUF and TBUF)	Move data from the R Buffer to the DRAM or from the DRAM to the T Buffer.	Section 3.2.25
CAP (Enumerated CSR Addressing)	CSR Access Proxy. Access to Fast write CSRs.	Section 3.2.18
CAP (Calculating Addressing)	CSR Access Proxy. Access to Global CSRs and other Microengine CSRs.	Section 3.2.19
CAP (Reflect)	Move data between different Microengines Transfer Registers.	Section 3.2.20
CTX_ARB	Perform context swap and wake on event.	Section 3.2.22
HALT	Puts the current thread to sleep without waking up any other thread and interrupts the Intel XScale® core.	Section 3.2.27
HASH	Issue a request to Hash Unit to perform an n-bit hash.	Section 3.2.28
MSF	Issue a request to the Media Switch Fabric.	Section 3.2.37
PCI	Issue a request to the PCI Unit.	Section 3.2.40
SCRATCH (Read and Write)	Move data between the Microengines and scratch memory.	Section 3.2.43
SCRATCH (Atomic Operations)	Issue a reference to perform an atomic operation on data in scratch memory	Section 3.2.44
SCRATCH (Ring Operations)	Issue Scratch Ring put or get commands to scratch memory.	Section 3.2.45
SRAM (Read and Write)	Move data between the Microengines and SRAM	Section 3.2.46
SRAM (Atomic Operations)	Issue a reference to perform an atomic operation on data in SRAM.	Section 3.2.47
SRAM (CSR)	Issue a reference to read or write the SRAM channel Control and Status Registers.	Section 3.2.48
SRAM (Read Queue Descriptor)	Issue a memory reference to an SRAM Channel to read the Queue Descriptor into the SRAM.	Section 3.2.49
SRAM (Write Queue Descriptor)	Issue a memory reference to an SRAM Channel to move data from the Queue Descriptor into SRAM.	Section 3.2.50
SRAM (Enqueue)	SRAM enqueue.	Section 3.2.51
SRAM (Dequeue)	SRAM dequeue.	Section 3.2.52
SRAM (Ring Operations)	Issue an SRAM Ring put or get command to the SRAM.	Section 3.2.53
SRAM (Journal Operations)	Issue an SRAM Journal command to the SRAM Channel.	Section 3.2.54
MEv2 CAM Instructions		
CAM_CLEAR	Clears all entries in the Microengine CAM.	Section 3.2.12
CAM_WRITE_STATE	Write the value for the State bits into the specified Microengine CAM entry.	Section 3.2.17

Table 3-1. Summary of Microengine Instructions (Continued)

Instruction	Description	Section
CAM_READ_TAG	Read the tag for the specified CAM entry into the destination register.	Section 3.2.13
CAM_READ_STATE	Read the State bits for the specified CAM entry.	Section 3.2.15
CAM_LOOKUP	Search the Microengine CAM for the tag value.	Section 3.2.13
CAM_WRITE	Write a value to the tag for the specified CAM entry.	Section 3.2.16

3.1 Instruction Syntax

This section describes the syntax for the MEv2 instruction set.

3.1.1 Restricted and Unrestricted Src and Dest Operands

Many instructions specify one or two source operands, and/or one destination operand. The choice of source and destination addressing is specified for each of those instructions as either Restricted or Unrestricted. (Table 3-2). Unrestricted indicates that there are enough bit in the instruction encoding to support all the possible options for source and destination addressing while restricted indicates that there are less bits and only a subset of the unrestricted options are possible

Table 3-2. Source/Destination Choices for Addressing Modes

Register Type	Unrestricted Addressing	Restricted Addressing
GPR	Context-Relative Absolute	Context-Relative
S Transfer	Context-Relative Indexed	Context-Relative Indexed
D Transfer	Context-Relative Indexed	Context-Relative Indexed
Neighbor	Context-Relative Indexed	Indexed
Local Memory	Indexed Offset (0-15)	Offset (0-7) Note 5
Other	immed no dest "--"	immed no dest "--"

NOTES:

- 8-bit Immediate data can be specified as a source for both Unrestricted and Restricted with the exception of I/O instructions which uses 7-bit immediate data.
- When a Transfer Registers is used as source, the Read Transfer Register is used. When a Transfer Register is used as a destination, the Write Transfer Register is written
- The notation "--" when used as a source means no source, used for single operand instructions
- The notation "--" when used as a destination means no destination; typically used to set Condition Codes.
- The post increment and post decrement operations are considered index mode operations and therefore can not be performed in the restricted address mode.

All instructions that specify a destination can also specify "No Destination" (using the -- notation). This is typically used to set ALU condition codes for branches without modifying any registers.

3.1.1.1 Two Source Operand Selection Rules

For instruction types that specify two source operands, there are some restrictions as to which registers can be used. Table 3-3 shows the legal combinations of A and B operands. For example, the “No” in column 4, row 4 means that two D Transfer Registers can not be used as source operands in an instruction.

Table 3-3. Legal Combinations of Source Operands

A Source	B Source						
	GPR (A Bank)	GPR (B Bank)	S Transfer	D Transfer	Neighbor	Local Memory	Immed
GPR (A Bank)	No	Yes	Yes	Yes	Yes	Yes	Yes
GPR (B Bank)	Yes	No	Yes	Yes	Yes	Yes	Yes
S Transfer	Yes	Yes	No	No	No	Yes	Yes
D Transfer	Yes	Yes	No	No	No	Yes	Yes
Neighbor	Yes	Yes	No	No	No	Yes	Yes
Local Memory	Yes	Yes	Yes	Yes	Yes	No	Yes
Immediate	Yes	Yes	Yes	Yes	Yes	Yes	No

The above can be summarized by three rules:

- Can’t use the same source as both A and B operands.
- Can’t use any of S Transfer, D Transfer and Neighbor as both A and B operands.
- Can’t use immediate as both A and B operands.

3.1.2 I/O Instruction Format

The MEv2 instruction set contains a class of instructions that are referred to as I/O instructions. The I/O instructions include:

- CAP
- DRAM
- MSF
- PCI
- SCRATCH
- SRAM
- HASH

All these instructions generate a command that is issued to an SRAM Controller, DRAM controller, SHAC, PCI, or MSF and all have the common properties described in this section.

3.1.2.1 Source Operands (src_op1, src_op2)

Two source operand parameters are required to form an address for the sram, dram, msf, pci, scratch, and cap (calculated addressing) I/O instructions. The address is formed by the src_op1 + src_op2. When the instruction specifies a reference count (ref_cnt) greater than 1, the address specifies the first address of the burst.

The rules specified in Table 3-3 are imposed when specifying src_op1 and src_op2.

3.1.2.2 Reference Count (ref_cnt)

The reference count specifies the burst size of the reference. The reference count specified in the instruction is always 1 to 8 and counts from 1 to 16 can be specified using an indirect reference. How this number is interpreted is dependent on the I/O instruction and the command. Table 3-4 shows the how the count value is interpreted for the I/O instructions. When Ref_cnt is interpreted as 8-bytes words, and the data is moved to or from the ME transfer registers, two transfer registers are used for each transfer.

Table 3-4. Reference Count Sizes

Ref_cnt Increments	I/O Instructions (Commands)
4 byte word	CAP, PCI, SCRATCH, SRAM, DRAM (CSR), MSF (read, write)
8 byte word	DRAM(read, write, tbuf_wr, rbuf_rd, MSF (read64, write64))

3.1.2.3 Optional Tokens (opt_tok)

Most of the I/O instructions support optional tokens. The common tokens are defined in this section and each instruction definition lists the specific token it supports. Tokens specific to an instruction are defined in the instruction definition.

Table 3-5. I/O Command Token Descriptions

Token	Description
ctx_swap[sig_name]	Put the thread to sleep and wake it when the specified signal event (sig_name) is asserted. Not used with sig_done.
sig_done[sig_name]	Signal the ME when the reference completes using the specified signal (sig_name) and continues executing. Not used with ctx_swap.
defer[m]	Execute "m" instruction(s) following this instruction before branching or switching contexts. The value of "m" is dependent on the instruction and can range from 1 to 3. Only used with ctx_swap.
indirect_ref	Use the ALU output of the previous instruction to redefine the instruction. The format of the ALU output data of the previous instruction is dependent on the I/O instruction and is defined in the individual I/O instruction definitions.
ignore_data_error	Signal the ME thread upon completion if a data error occurs on a read operation. If this token is not specified, the ME thread is not signalled.
ind_targets[me1, me2, ...]	Informs the assembler that the target of an IO instruction is in a different ME (me1, me2, ...). As a result, any generated signals and transfer registers are considered to be remote and are looked up in the targets. to use this optional token, the indirect reference must override the ME number.

3.1.2.3.1 Indirect References

The I/O instructions supports an indirect reference optional token. When the indirect_ref optional token is specified, the output of the ALU from the previous instruction is used to modify one or more parameters within an instruction. The format of the indirect data is specified in the instruction definition. All Reserved fields noted as RES, must be written to 0 else unpredictable behavior may result.

An override bit is provided on a per-data field basis to indicate which parameter should be modified. If OV for the field is a '0', the field value is not used (and its field value has no effect). If OV for the field is a '1' the field is used in place of the information supplied in the instruction.

The format for the `cap[fast_wr]` command using the `alu` keyword in the data field differs from the other formats. In this case the full 32-bits of the ALU output are used as the `fast_wr` data and there are no override bits.

The following notes pertain to indirect references in general.

- ME Number, Context Number, and Transfer Register Number can all be overridden separately.
- There are two aspects of the transfer that are affected when ME Number is overridden—which ME's transfer register(s) is the source/destination of data, and which ME is signaled.
- When the Transfer Register Number is overridden it is treated as an absolute register number. If Context Number is also overridden it determines where the signal is delivered, but not the Transfer Register Number.
- When the Transfer Register Number is not overridden, and Context Number is, Context Number specifies where the signal is delivered, as well as the Context Number of the transfer register.

3.1.2.3.2 Changing the Ref_Cnt using Indirect References

There is an issue that concerns the assembler when redefining the `ref_cnt` (reference count) field using an Indirect References. The issue is that since the indirect reference data is specified at runtime, the assembler's register allocator has no way of knowing how many registers will be used for the I/O instruction. To handle this, the reference count field in the instruction parameter list can take values of the form:

`max_nn` where `nn` is a decimal constant between 1 and 16.

The `max_nn` keyword is only valid if the `indirect_ref` optional token is specified. It allows the programmer to manually indicate to the assembler the maximum number of 32-bit/64-bit words that will be specified indirectly.

Note that the register allocator needs to know the maximum number of registers used so that it can allocate the correct number of registers. A program that species a maximum number and uses fewer registers will still work correctly. However, a program that species a maximum number and uses more register will not work correctly and is considered a programming error.

Another issue involves the fact that the maximum reference count that can be encoded in the instruction is 8, whereas the maximum reference count that can be specified indirectly is 16. A count value must be specified in the instruction (stored in the control store) during assembly time regardless of whether or not the `indirect_ref` optional token is specified, and since the instruction only supports a three bit count field, the assembler will use the lesser of `nn` and 8.

Also, it is possible to specify the `indirect_ref` and not change the `ref_cnt` value in the indirect reference data, however this is considered a programming error and will not be detected by the assembler.

Note that if the maximum count is less than or equal to 8, then the programmer can still use `max_nn` although there would be no difference if they used `nn` directly.

Additionally, the reference count field can take the value "max". This is equivalent to "max_nn", except that the count is taken by counting how many registers follow the given register in `.xfer_order` order.

The following are valid examples of this:

```
sram[read, $xfer, a1, a2, 6], indirect_ref
```

```
sram[read, $xfer, a1, a2, max_6], indirect_ref
sram[read, $xfer, a1, a1, max_12], indirect_ref
```

In these example, the reference counts encoded in the instruction are: 6, 6, 8.

The following are invalid examples:

```
sram[read, $xfer, a1, a2, 12], indirect_ref
sram[read, $xfer, a1, a1, max_12]
```

The problem with the first of these is that “12” is too large to be encoded in the instruction. The problem with the second is that “indirect_ref” is not specified.

The REF CNT field in the indirect reference specifies the number of words to be transferred to or from a set of Transfer Registers, where words are 8 bytes in length for DRAM transfers and 4 bytes in length for other units transfers. Since the Transfer Registers are selected by specifying the starting Transfer Register and a reference count it is possible to specify a REF CNT and the transfer register that the results in a data transfer that extends beyond a thread’s relatively addressed Transfer Register set. The programmer can ensure that a Transfer Register does not extend into the next thread by specifying a contiguous series of transfer registers using the.xfer order directive, specifying the first register in the series, and ensuring that the REF_CNT does not exceed the number of registers specified in the series.

3.1.2.3.3 Indirect References to another ME

Through an indirect reference, one ME can direct the I/O operation to another ME. For example, ME-0 can issue an sram/read operation and have the read results (and signal) go to ME-1. To do this, the assembler must be informed as to the actual target ME. Note that it is also possible to have multiple targets, in the case where the actual ME is selected at run-time. To do this, add the optional token “ind_targets[me1, me2, ...]” to the I/O instruction. This is only valid if the indirect-ref token is also specified.

When the “ind_targets” token is provided, then it is assumed that the indirect reference is going to override the ME. This causes two things to happen: Any generated signal is considered to be remote and is looked up in the targets (if there are multiple targets, then it must have the same address in all specified MEs). Secondly, the register specified in the I/O operation is assumed to be remote and is similarly looked up. Even if the transfer register address is being overridden, a valid remote register must be specified. This is needed to determine whether the remote register is an S-xfer or D-xfer. If the transfer register address is not overridden, the transfer register used is for the same context as the issuing context. If the issuing ME is running an odd context and the remote ME is in 4-context mode, then the results are unpredictable. On the other hand, if the transfer register address is overridden, then the context can be explicitly specified.

When the ind_targets token is provided, then there is no need to “complete” the I/O operation because the I/O operation is not referencing any of the local transfer registers or signals. As such, it is not allowed to use the ctx_swap token with targets.

It is required that the number of contexts in the remote and local ME are the same. If the number of contexts in the remote ME is not the same as in the local ME, the behavior is undefined

For example, if ME-0 wanted to issue an SRAM read to ME-1, this would be coded:

```
ME-0:
.reg remote $x
.sig remote sig
#define ME 1
alu[--,--,b,(0x20 | ME), <<26]
```

```
sram[read, $x, addr1, addr2, 1], sig_done[sig], ind_targets[ME]
; no I/O operation to complete

ME-1:
.reg visible $x
.sig visible sig

ctx_arb[sig]
; $x now has the sram value
```

3.1.2.4 Event Signals

The I/O instructions provide the option of having the ME signaled when data is pushed or pulled from the transfer register. The I/O instruction/commands can be classified into types based on the number of signals that can be requested:

- Single signal provided when data is pulled from the write transfer register
- Single signal provided when data is pushed to the read transfer register
- Two signals provided one for each case above
- No signal provide because data is neither pulled or pushed

There is an additional special type that is required only when memory channels are interleaved as is the case for the IXP2800 DRAM channels. Although there is only one DRAM channel on the IXP2400 network processor, two signals are used as well.

- Two signals are provided. If a burst splits between two DRAM channels, each channel will generate a signal. If the burst does not split between two channels, a single channel will generate both signals.

Signals are generated and tested using the instructions listed in [Table 3-6](#).

Table 3-6. Instructions and Optional Tokens that use Signals (Sheet 1 of 2)

Option		Description
Instruction	Optional Token	
Any I/O Instruction	sig_done[sig_name]	1.Request Signal 2.Thread continues to execute
	ctx_swap[sig_name]	1.Request Signal 2.Put thread to sleep 3.Wake when signaled and clear signal. Although this performs the same functionality as a I/O instruction with a sig_done optional token and a ctx_arb instruction it still only takes a single cycles to execute the I/O instruction. Note: ctx_swap is not used with instructions that require two signals

Table 3-6. Instructions and Optional Tokens that use Signals (Sheet 2 of 2)

Option		Description
Instruction	Optional Token	
br_signal br_!signal		1. Test Signal 2. Branch appropriately and clear signal if set. See Note 1
ctx_arb		1. Put thread to sleep 2. Wake when signaled and clear signal. The ctx_arb instruction supports waking on a list of signals. See Note 1
Note 1: These instructions use signals requested by an I/O instruction that specifies the sig_done[sig_name] optional token.		

The assembler supports a register allocator that manages the allocation of signals for the programmer. The programmer uses symbolic names for signals rather than explicit numbers, and the assembler assigns one or more signal numbers from a pool of fifteen signal (1-15). The hardware requires that I/O instructions which require two signals be assigned consecutive signals and that the first signal is an even number. The assembler hides this restriction from the programmer except for the special case of the br_signal and br_!signal instructions. These instructions can only test one signal at a time so the programmer must provide two branch instructions to test both signals. The first signal is specified by the name, the second signal is specified as the name with a "+1" suffix (example: signal_name+1). Table 3-7 lists the number of signals generated for each instruction and instruction command.

This document uses the following terminology in the instruction set definitions Section 3.2) to refer to signals names that are mapped to two signals:

- sig_name: One signal is provided
- sig_name2: Two signals are provided

For example ctx_swap[sig_name] or ctx_swap[sig_name2].

Table 3-7. Signal Restrictions for each I/O Instruction [command] (Sheet 1 of 3)

Instruction	Command	Valid Signals	Comments
DRAM	read write tbuf_wr rbuf_rd	2 to 15 Pair beginning with even signal numbers	IXP2800: If a bursts splits between two DRAM channels, each channel will generate a signal. If the burst does not split between two channels a single channel will generate both signals. IXP2400: If a bursts splits between two DRAM banks, the controller will generate a separate signal when each bank operation is complete. If the burst does not split between two banks, the controller will generate both signals when the single bank operation is complete.
CAP	read write	1 to 15	
	fast_wr	Not Used	
hash_48 hash_64 hash_128	not applicable	2 to 15 Pair beginning with even signal numbers	The "name" signal will be even and it indicates that data has been pulled from the transfer register, while "name" + 1 signal indicates that the hash result has been returned to the transfer register.

Table 3-7. Signal Restrictions for each I/O Instruction [command] (Sheet 2 of 3)

Instruction	Command	Valid Signals	Comments
MSF	read, write, read64, write64	1 to 15	
	fast_wr	Not Used	
PCI	read	1 to 15	
	write	1 to 15	
SCRATCH	read, write	1 to 15	
	incr, decr	Not Used	
	swap, test_and_set, test_and_clr, test_and_add, test_and_sub	2 to 15 Pair beginning with even signal numbers	The "name" signal will be even and it indicates that data has been pulled from the transfer register, while "name" + 1 signal indicates that the swapped or pre-modified data has been returned to the transfer register.
	test_and_incr, test_and_decr, set, clr, add, sub	1 to 15	
	get, put	1 to 15	

Table 3-7. Signal Restrictions for each I/O Instruction [command] (Sheet 3 of 3)

Instruction	Command	Valid Signals	Comments
SRAM	read, write	1 to 15	
	test_and_incr, test_and_decr set, clr, add,	1 to 15	
	Regular version: test_and_set, test_and_clr, test_and_add, swap	2 to 15 Pair beginning with even signal numbers	The "name" signal will be even and it indicates that data has been pulled from the transfer register, while "name" + 1 signal indicates that the swapped or pre-modified data has been returned to the transfer register.
	No-pull version (IXP28xxRev B): test_and_set, test_and_clr, test_and_add, swap	1 to 15	
	incr, decr	Not Used	
	csr_rd, csr_wr	1 to 15	
	rd_qdesc_head, rd_qdesc_tail,	1 to 15	
	wr_qdesc, wr_qdesc_count	Not Used	
	enqueue enqueue_tail	Not Used	
	dequeue	1 to 15	
	get	1 to 15	
	put	2 to 14 Pair beginning with even signal numbers	The "name" signal will be even and it indicates that data has been pulled from the transfer register, while "name" + 1 signal indicates that the ring status word has been returned to the transfer register.
	journal	1 to 15	
fast_journal	Not Used		

3.1.3 Condition Codes

There are four Condition Codes, which are modified by some instructions, and are used for branch instructions. The Condition Codes are Negative (N), Zero (Z), Carry Out (C), and Overflow (V). Condition Code operation for all instructions is shown with the instruction description.

Note: There is only one set of Condition Codes, not a set per Context. Therefore a Context can not count on the Condition Code values before a context swap being available when it resumes from that

swap, because other Contexts will change them. However, if only one Context is enabled, Condition Codes will not be changed when that Context swaps.

3.1.4 Branch Defer (defer[n])

Any instruction that makes a branch decision may cause one or more instructions in the execution pipeline to be aborted. An instruction that makes a branch decision does so based on the result of an operation that occurs in either the P1, P2, P3, or P4 instruction pipeline stage. The specific pipeline stage where the decision is made depends on the instruction.

The purpose of a deferred branch is to reduce or eliminate aborted instructions in the execution pipeline. In a deferred branch, an instruction that follows a branch decision is allowed to execute before the branch takes effect (i.e., the effect of the branch is “deferred” in time). If useful work can be found to fill the wasted cycles after the branch instruction, the branch latency can be hidden.

Deferred branches are supported using the “defer” optional token within an instruction. The Assembler’s optimizer can automatically insert the deferred token and re-arrange the instructions or the programmer can do it manually.

Examples: Defer[n] branch taken

```
alu[temp2,temp1,-,temp3] ; always executed
bgt[branch_taken_label#], defer[3] ; (assume branch taken)
alu[temp,temp,+,temp2] ; always executed (would have been aborted)
alu[temp,--,B,temp2] ; always executed (would have been aborted)
alu[--,temp,-,temp3] ; always executed (would have been aborted)
alu[temp,--,B,temp1] ; not executed (branch was taken)
```

Table 3-8 shows the legal defer options for all branch types. The no defer column gives the number of instruction cycles lost on a taken branch if no defer token is used. Each deferred instruction reduces that penalty by one. The defer [n] columns indicate if that defer amount is allowed for the given branch type.

Table 3-8. Branch Defer Summary

Branch Type	Penalty on Branch Taken with no defer	defer allowed?		
		defer [1]	defer [2]	defer [3]
Unconditional Branch	2	Yes	Yes	No
Context Swap (note)	2	Yes	Yes	No
Conditional Branch when condition was set earlier than prior instruction	2	Yes	Yes	No
Conditional Branch when condition was set in prior instruction	3	Yes	Yes	Yes
Jump	4	Yes	Yes	Yes

Table 3-8. Branch Defer Summary

RTN	4	Yes	Yes	Yes
Branch on Input State	4	Yes	Yes	Yes
Branch on Signal	4	Yes	Yes	Yes
BR_Byte	4	Yes	Yes	Yes
BR_Bit	4	Yes	Yes	Yes
Note: Context Swap applies to I/O reference instructions that support the ctx_swap optional token and the ctx_arb instruction. These instructions are special cases of branch instructions since they put a thread to sleep and branch to the instruction for the next context.				

3.1.5 Coding Restrictions

This section lists coding restrictions.

Note: This section is not complete at this time. A more complete list will be added in future revs.

3.1.5.1 Branch or I/O Command in Defer Slot

A branch, jump, context swap or any IO command cannot be placed in a defer slot.

Example 3-1.

```
br[A#], defer[1]
br[B#]           ; illegal
```

Example 3-2.

```
ctx_arb[sig_name], defer[1]
jump[offset, A#]; illegal
```

Example 3-3.

```
sram[write, $data, base, offset, 1], ctx_swap[sig_name], defer[1]
ctx_arb[sig_name] ; illegal
```

Example 3-4.

```
alu[i, i, +, 1]
BEQ[A#], defer[2]
alu[i, i, +, 1]
BLT[B#]           ; illegal
```

Example 3-5.

```
jump[offset, A#], defer[3]
alu[i, i, +, 1]
alu[i, i, +, 1]
ctx_arb[1]        ; illegal
```

Example 3-6.

```
ctx_arb[voluntary], defer[1]
sram[read, $data, base, offset, 1], sig_done; illegal
```

Example 3-7.

```
br[A#], defer[1]
sram[read, $data, base, offset, 1]; illegal
```

3.1.5.2 Condition Codes after Swap

Condition Codes are not stored during a context swap.

Example 3-1.

```
ctx_arb[voluntary]
BEQ[A#] ; illegal
```

Example 3-2.

```
ctx_arb[1]
BEQ[A#] ; illegal
```

Example 3-3.

```
sram[write, $data, base, offset, 1], ctx_swap[sig_1]
BEQ[A#] ; illegal
```

Example 3-4.

```
sram[write, $data, base, offset, 1], ctx_swap[sig_1]
br=byte[i,1,0x0, A#]; legal
```

3.1.5.3 CAM after Conditional P3 Branch

An operation that modifies any state of the CAM cannot immediately follow a conditional P3 branch. CAM state includes Tags, State bits, and LRU/MRU logic.

(Note that a CAM can be placed in the defer slot of a Conditional P3 branch.)

Example 3-1.

```
br=byte[i,1,0x0, A#]; p3 branch
CAM_Write_State[entry , 0x2]; illegal
```

Example 3-2.

```
br=byte[i,1,0x0, A#]; p3 branch
alu[i,i,+1]
CAM_Write_State[entry , 0x2]; legal
```

Example 3-3.

```
br=byte[i,1,0x0, A#], defer[1]; p3 branch
CAM_Write_State[entry , 0x2]; legal
```

Example 3-4.

```
br_signal [1, A#] ; p3 branch
Lookup_CAM[status, lookup_value]; illegal
```

Example 3-5.

```
jump [offset, A#] ; p3 jump
Lookup_CAM[status, lookup_value]; legal
```

3.1.5.4 Dram with Swap

A dram command cannot use a ctx_swap token.

Example 3-1.

```
dram[read, $$val, base, offset, 1], ctx_swap[sig_4]; illegal
```

Example 3-2.

```
dram[read, $$val, base, offset, 1], sig_done[sig_4] ; legal
ctx_arb[sig_4, sig_6], and ; legal
```

Example 3-3.

```
sram[read, $val, base, offset, 1], ctx_swap[sig_4]; legal
```

3.1.5.5 BCC after Conditional P3 branch

Can not be execute a BCC instruction immediately after a branch conditional instruction that makes a branch decision in ME P3 pipe stage.

Example 3-1.

```
br=byte[i,1,0x0, A#]; p3 branch
BNE[A#] ; illegal
```

Example 3-2.

```
br_bset[reg, bit, A#]; p3 branch
BLT[A#] ; illegal
```

Example 3-3.

```
br!signal[1,A#]; p3 branch  
BEQ[A#]           ; illegal
```

Example 3-4.

```
br_inp_state[state, A#]; p3 branch  
BGT[A#]           ; illegal
```

Example 3-5.

```
jump[offset, A#]; p3 branch  
BLT[A#]           ; legal
```

Example 3-6.

```
br=byte[i,0,0x0, A#]; p3 branch  
br=byte[i,1,0x0, B#]; legal  
br=byte[i,2,0x0, C#]; legal  
br=byte[i,3,0x0, D#]; legal
```

Example 3-7.

```
br=byte[i,1,0x0, A#]; p3 branch  
alu[i,i,+,1]  
BEQ[A#]           ; legal
```

Example 3-8.

```
br=byte[i,1,0x0, A#],defer[1]; p3 branch  
alu[i,i,+,1]  
BEQ[A#]           ; legal
```

Example 3-9.

```
alu[i,i,+,1]  
br=byte[i,1,0x0, A#]; p3 branch  
nop  
BEQ[A#]           ; legal
```

3.1.5.6 LOCAL_CSR_RD cannot be in last defer slot.

A LOCAL_CSR_RD cannot be placed in the last defer slot of a branch, ctx_arb, ctx_swap, jump, or rtn.

Example 3-1.

```
ctx_arb[1], defer[1]
local_csr_rd[ACTIVE_LM_ADDR_0]; illegal
```

Example 3-2.

```
ctx_arb[1], defer[2]
local_csr_rd[ACTIVE_LM_ADDR_0]; legal
immed[temp, 0]
```

Example 3-3.

```
br[label#] defer[2]
alu[i,i,+,1]
local_csr_rd[NN_GET]; illegal
```

Example 3-4.

```
rtn[address], defer[3]
alu[i,i,+,1]
local_csr_rd[Active_CTX_sts]; legal
immed[temp,0]
```

Example 3-5.

```
rtn[address], defer[3]
alu[i,i,+,1]
nop
local_csr_rd[Active_CTX_sts]; illegal
```

3.1.5.7 LOCAL_CSR_WR to ACTIVE_LM_ADDR, or CAM_LOOKUP

LOCAL_CSR_WR to ACTIVE_LM_ADDR, or CAM_LOOKUP (with lm_addr#[num] token) placed in the defer slots of a CTX_ARB instruction (or command instruction with ctx_swap token)

Example 3-1.

```
ctx_arb[1], defer[1]
local_csr_wr[ACTIVE_LM_ADDR_0, value]; illegal
```

Example 3-2.

```
ctx_arb[1]
local_csr_wr[ACTIVE_LM_ADDR_0, value]; legal
```

Example 3-3.

```
ctx_arb[1], defer[1]
local_csr_wr[ACTIVE_LM_ADDR_1_BYTE_INDEX, value]; illegal
```

Example 3-4.

```
ctx_arb[1], defer[1]
local_csr_wr[INDIRECT_LM_ADDR_1, value]; legal because not ACTIVE_LM_ADDR
```

Example 3-5.

```
ctx_arb[1], defer[2]
alu[i,i,+1]
local_csr_wr[ACTIVE_LM_ADDR_0, value]; illegal
```

Example 3-6.

```
ctx_arb[1], defer[1]
cam_lookup[@gpr_a,@gpr_b], lm_addr0[value] ; illegal
```

3.1.5.8 LOCAL_CSR_RD must be followed by an IMMED op

LOCAL_CSR_WR to ACTIVE_LM_ADDR or CAM_LOOKUP (with lm_addr#[num] token) placed in the defer slots of a CTX_ARB instruction (or command instruction with ctx_swap token)

Example 3-1.

```
local_csr_rd[NN_get]
local_csr_wr[NN_put, 0x2] ; illegal
```

Example 3-2.

```
local_csr_rd[ACTIVE_LM_ADDR_0]
local_csr_rd[NN_PUT] ; illegal
```

Example 3-3.

```
local_csr_rd[ACTIVE_CTX_STS]
alu[i,i,+1] ; illegal
```

Example 3-4.

```
local_csr_rd[ACTIVE_CTX_STS]
immed[temp,0] ; legal
```

3.1.5.9 I/O Command Op after LOCAL_CSR_WR

If an I/O instruction (i.e. cap, msf, scratch, dram, pci, sram, or hash) is issued within 3 instruction cycles of a local_csr_write, the CSR should not be used or read within the 3 cycle window.

Example 3-1.

```
local_csr_wr[ active_lm_addr_1, reg_a18]
dram[write , $$dxfer15, *1$index1, offset, ref_cnt], sig_done[sig_2] ;illegal
```

Example 3-2.

```
local_csr_wr[ active_lm_addr_0, reg_a18]
immed_w1[ reg_a31, 0 ]
sram[read , $sxfer15, *1$index0, offset, ref_cnt], ctx_swap[sig_2]; illegal
```

Example 3-3.

```
local_csr_wr[ active_lm_addr_0, reg_a18]
alu [ reg_a1, --, B, reg_a2 ]
immed_w0[ reg_a2, 0 ]
sram[read , $sxfer15, *1$index0, offset, ref_cnt], sig_done[sig_2]; illegal
```

Example 3-4.

```
local_csr_wr[ t_index, reg_a18]
immed [ reg_a1, 0 ]
sram[read , $sxfer15, reg_a1, offset, ref_cnt], sig_done[sig_2]
alu [*$index++, --, B, reg_a10]; illegal
```

Example 3-5.

```
local_csr_wr[ t_index, reg_a18]
immed [ reg_a1, 0 ]
sram[read , $sxfer15, *$index, offset, ref_cnt], sig_done[sig_2]; illegal
```

Example 3-6.

```
local_csr_wr[ t_index, reg_a18]
immed [ reg_a1, 0 ]
sram[read , $sxfer15, *1$index0, offset, ref_cnt], sig_done[sig_2]
local_csr_rd [t_index] ; illegal
immed [tmp, --]
```

Example 3-7.

```
local_csr_wr[ active_lm_addr_0, reg_a18]
nop;
nop;
nop;
sram[read , $sxfer15, *1$index0, offset, ref_cnt], sig_done[sig_2]; legal
```

Example 3-8.

```
local_csr_wr[ active_lm_addr_0, reg_a18]
alu [ reg_a1, *1$index0, + reg_b1 ] ; legal
immed_w0[ *1$index0, 0 ] ; legal
immed_w1[ *1$index0, 1 ] ; legal
sram[read , $sxfer15, *1$index0, offset, ref_cnt], sig_done[sig_2]; legal
```

Example 3-9.

```
local_csr_wr[ t_index, reg_a18]
immed [ reg_a1, 0 ]
sram[read , $sxfer15, reg_a1, offset, ref_cnt], sig_done[sig_2]
nop
alu [*$index++, --, B, reg_a10] ; legal
```

3.1.5.10 LOCAL_CSR_WR to CTX_WAKEUP_EVENTS

LOCAL_CSR_WR to CTX_WAKEUP_EVENTS should be directly followed by a CTX_ARB[--] or should be inside the defer shadow of a CTX_ARB[--]

Example 3-1.

```
local_csr_wr[ACTIVE_CTX_WAKEUP_EVENTS, value]
ctx_arb[--]; legal
```

Example 3-2.

```
ctx_arb[--], defer[1]
local_csr_wr[ACTIVE_CTX_WAKEUP_EVENTS, value]; legal
```

Example 3-3.

```
local_csr_wr[ACTIVE_CTX_WAKEUP_EVENTS, value]
alu[temp, temp, +, 1]; unrelated instruction
ctx_arb[--]; illegal
```

3.1.6 MEv2 Permitted Coding Sequences

The following coding sequences are permitted on MEv2. They are illustrated for clarification, as they may have been illegal in the IXP1200 MEv1 and/or illegal in earlier versions of MEv2.

3.1.6.1 Swap after P3 Branch

P3 branches (including jumps) CAN be followed by a context swap.

Example 3-1. Example 1

```
br=byte[i,1,0x0, A#]; p3 branch
ctx_arb[1] ; legal
```

Example 3-2. Example 2

```
jump[offset, A#]; p3 branch
ctx_arb[1] ; legal
```

Example 3-3. Example 3

```
br!signal[1,A#]; p3 branch
ctx_arb[1] ; legal
```

Example 3-4. Example 4

```
br_bset[reg, bit, A#]; p3 branch
ctx_arb[1] ; legal
```

3.1.6.2 Memory Command after P3 Branch

P3 branches (including jumps) CAN be followed by a command memory op.

Example 3-1. Example 1

```
br_bset[reg, bit, A#]; p3 branch
sram[write, $data, base, offset, 1], ctx_swap[sig_1]; legal
```

Example 3-2. Example 1

```
jump[offset, A#] ; p3 branch
dram[write, $data, base, offset, 2]; legal
```

3.1.6.3 Swap after Voluntary Swap.

A context swap instruction CAN immediately follow a voluntary context swap.

Example 3-1. Example 1

```
ctx_arb[1]
ctx_arb[2]           ; legal
ctx_arb[voluntary]  ; legal
ctx_arb[3]           ; legal
```

Example 3-2. Example 2

```
ctx_arb[1]
ctx_arb[2]           ; legal
ctx_arb[voluntary]  ; legal
alu[i,i,+,1]
ctx_arb[3]           ; legal
```

Example 3-3. Example 3

```
ctx_arb[voluntary]
sram[write, $data, base, offset, 1], ctx_swap[sig_1]; legal
```

3.1.6.4 A LOCAL_CSR_WR in defer slot

A LOCAL_CSR_WR can be placed in the last defer slot of a branch, ctx_arb, ctx_swap, jump, or rtn.

(NOTE: there is a separate rule whereby it is illegal to write the ACTIVE_LM_ADDR's in any defer slot. see [Table 3.1.5.7](#). Also, some CSR's like, USTORE_ERROR_STATUS are read-only.)

Example 3-1. Example 1

```
ctx_arb[1], defer[1]
local_csr_wr[NN_get, value]; legal
```

Example 3-2. Example 2

```
ctx_arb[1], defer[2]
Alu[i,i,+,1]
local_csr_wr[NN_put, 0x0]; legal
```

Example 3-3. Example 3

```
alu[i,i,+,1]
BEQ[label#], defer[1]
local_csr_wr[Same_ME_Signal,sig_value]; legal
```

Example 3-4. Example 4

```
br[label#], defer[2]
alu[i,i,+,1]
local_csr_wr[Same_ME_Signal,sig_value]; legal
```

Example 3-5. Example 5

```
rtn[address], defer[3]
alu[i,i,+,1]
immed[tmp,2]
local_csr_wr[Active_ctx_future_count,fut_count]; legal
```

3.1.6.5 LOCAL_CSR_WR can be followed by a LOCAL_CSR_RD or LOCAL_CSR_WR.**Example 3-1. Example 1**

```
local_csr_wr[NN_get, value]
local_csr_wr[NN_put, 0x2]; legal
```

Example 3-2. Example 2

```
local_csr_wr[Same_ME_Signal,sig_value]
local_csr_rd[NN_put] ; legal
immed[tmp,0]
```

3.2 Instruction Set

3.2.1 ALU

Perform an ALU operation on one or two source operands and deposit the result into the destination register. Update all ALU condition codes according to the result of the operation.

Instruction Format

`alu[dest, A_op, alu_op, B_op]`

Parameter Descriptions

Parameter	Description	
dest	Unrestricted destination that gets written with the result of the operation.	
A_op, B_op	Unrestricted source operand.	
alu_op	ALU Operation	ALU operation Description
	+	A operand + B operand.
	+16	A operand + (0xFFFF & B). B operand truncated to the least significant 16 bits (upper 2 bytes zeroed).
	+8	A operand + (0xFF & B). B operand truncated to the least significant 8 bits (upper 3 bytes zeroed).
	+carry	A operand + B operand + previous carry-in (carry-in equals previous carry-out).
	-carry	A operand - B operand - inverse of carry
	-	A operand - B operand.
	B-A	B operand - A operand.
	B	B operand (A operand is ignored).
	~B	Inverted B operand (A operand is ignored).
	AND	A operand AND B operand (Bit-wise AND).
	~AND	Inverted A operand AND B operand (Bit-wise AND).
	AND~	Inverted B operand AND A operand (Bit-wise AND).
	OR	A operand OR B operand (Bit-wise OR).
XOR	A operand XOR B operand (Bit-wise exclusive OR).	

Condition Codes Affected

ALU Operation	N	Z	V	C
+	Result[31] == 1	Result[31:0] == 0	Set if a signed overflow occurs. Note 1	Carry Out from Adder[31]
+16				
+8				
+carry				
-carry				
-			Cleared	Cleared
B-A				
B				
~B				
AND				
~AND	Cleared	Cleared		
AND~				
OR				
XOR	Cleared	Cleared		

1. Logically equivalent to Carry from Adder[31] XOR Carry from Adder[30].

3.2.2 ALU_SHF

Perform an ALU operation on one or two operands and deposit the result into the destination register. The B operand is shifted or rotated prior to the ALU operation.

If the programmer uses the command ALU in the place of ALU_SHF, the assembler will replace it with ALU_SHF.

Instruction Format

```
alu_shf[dest, A_op, alu_op, B_op, B_op_shf]
```

Parameter Descriptions

Parameter	Description	
dest	Restricted destination that gets written with the result of the operation.	
A_op, B_op	Restricted source operand.	
alu_op	ALU Operation	ALU operation Description
	B	B operand (A operand is ignored).
	~B	Inverted B operand (A operand is ignored).
	AND	A operand AND B operand (Bit-wise AND).
	~AND	Inverted A operand AND B operand (Bit-wise AND).
	AND~	Inverted B operand AND A operand (Bit-wise AND).
	OR	A operand OR B operand (Bit-wise OR).
B_op_shf	Shift Operation	Shift operation Description
	<<n	Left shift n bits, where n = 1 through 31.
	<<indirect	Left shift by an amount specified in the lower 5 bits of the A operand of the previous instruction (the previous instruction must be one of the following ALU or ALU_SHF instructions -- A AND B, A AND ~B, A XOR B, A OR B). The lower 5 bits of the A operand should be n, where n is the desired left shift amount.
	>>n	Right shift n bits, where n = 1 through 31.
	>>indirect	Right shift by the amount specified in the lower 5 bits of the A operand of the previous instruction.
	<<rotn	Left rotate n bits, where n = 1 through 31.
	>>rotn	Right rotate n bits, where n = 1 through 31.
	NOTE: 1. Indirect rotates are not allowed. However, an indirect rotate can be emulated by copying the desired register value to be rotated into a register on the opposite bank and then performing a dbf_shf instruction. 2. There should be no spacing between the shift symbol (<< , >>) and the variable or keyword that follows it.	

Condition Codes Affected

ALU Operation	N	Z	V	C
B	Result[31] == 1	Result[31:0] == 0	Cleared	Cleared
~B				
AND				
~AND				
AND~				
OR				

3.2.3 ASR

Perform an arithmetic shift right on a register. This is a two-instruction sequence. The first instruction must be a shift and/or logical instruction that puts a value in the MSB (bit 31) of the result (this instruction must be one of the following ALU or ALU_SHF instructions -- B, ~B, AND, ~AND, AND~, OR, XOR). The ASR instruction shifts a source register right, replicating that MSB into vacated bit positions.

Instruction Format

```
asr[dest, src, shf_amt]
```

Parameter Descriptions

Parameter	Description	
dest	Restricted destination that gets written with the result of the operation.	
src	Restricted source operand.	
shf_amt	>>n	Shift the src operand right by n bits. Valid values of n are 1 to 31.
	>>indirect	Right shift by the amount specified in the lower 5 bits of the A operand of the previous instruction.

Condition Codes Affected

N	Z	V	C
Result[31] == 1	Result[31:0] == 0	Cleared	Cleared

Examples

Example 1: Sign extend the low byte of r0 which is 0x80. The result is 0xFFFF FF80.

```
immed_w0[r0,0x80]
immed_w1[r0,0x0]
alu_shf[r0, --, B, r0, <<24] ; bit 31 of result determines sign
asr[r0, r0, >>24]
```

Examples

Example 1: Sign extend the low byte of r0 which is 0x80 using run time shift value. The result is 0xFFFF FF80.

```
immed_w0[r0,0x80]
immed_w1[r0,0x0]
alu[r1, --, B, 24] ; Set up the shift amount
;
alu[--, r1, OR, 1] ; a_op of this instr is r1 = 24
alu_shf[r0, --, B, r0, <<indirect] ; Bit 31 of result determines sign
; Result = 0x8000 0000
alu[--, r1, OR, r0] ;a_op of this instr = r1 = 24
asr[r0, r0, >>indirect]; shift >> 24 & sign extend 0x8000 0000
; result is 0xffff ff80
```

3.2.4 BCC (BRANCH CONDITION CODE)

Branch to an instruction at a specified label based on the condition codes set by a previous instruction. The condition codes supported are Sign (N), Zero (Z), Overflow (V), and Carry (C) out.

Instruction Format

```
bcc[label#], opt_tok
```

Table 3-9 describes the supported Bcc instructions. The terms less, greater, and equal are used for comparison of signed numbers while higher, lower, and same are used for unsigned numbers. Subtracting two operands using the ALU instruction performs comparisons (for example A-B is equivalent to compare A to B). A Branch if Higher function can be performed by reversing the operands of the subtraction operation and using BLO and a Branch Lower or Same can be performed by reversing the operands of the subtraction operation and using BHS. Since there is more than one way to interpret a particular state of the Condition Codes, the assembler provides more than one mnemonic for some states.

Table 3-9. Branch on Condition Code Instructions

Mnemonic (Bcc)	Description	Condition Code	Data Type
BEQ	Br if equal	Z == 1	Both
BNE	Br if not equal	Z == 0	Both
BMI	Br sign set (minus)	N == 1	Signed
BPL	Br sign clear (plus)	N == 0	Signed
BCS	Br if carry set	C == 1	Both
BHS	Br if higher or same		Unsigned
BCC	Br if carry clear	C == 0	Both
BLO	Br if lower		Unsigned
BVS	Br if overflow set	V == 1	Signed
BVC	Br if overflow clear	V == 0	Signed
BGE	Br if greater or equal	N == V (XNOR)	Signed
BLT	Br if less than	N != V (XOR)	Signed
BGT	Br if greater than	(Z == 0) AND (N == V)	Signed
BLE	Br if less than or equal	(Z == 1) OR (N != V)	Signed

Parameter Descriptions

Parameter	Description
label#	Symbolic label corresponding to the address of an instruction
opt_tok	defer[n] (n= 1 to 3) refer to Section 3.1.4

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.5 BR

Branch unconditionally.

Instruction Format

<code>br[label#], opt_tok</code>

Parameter Descriptions

Parameter	Description
<code>label#</code>	Symbolic label corresponding to the address of an instruction
<code>opt_tok</code>	<code>defer[n]</code> (n= 1 to 2) refer to Section 3.1.4

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.6 BR_BCLR, BR_BSET

Branch to the instruction at the specified label when the specified bit of the register is clear or set.

Instruction Format

<code>br_bclr[reg, bit_position, label#], opt_tok</code>
<code>br_bset[reg, bit_position, label#], opt_tok</code>

Parameter Descriptions

Parameter	Description
<code>reg</code>	Restricted source operand.
<code>bit_position</code>	A number specifying a bit position in a 32-bit word. Bit 0 is the least significant bit. Valid <code>bit_position</code> values are 0 through 31.
<code>label#</code>	Symbolic label corresponding to the address of an instruction
<code>opt_tok</code>	<code>defer[n]</code> (n= 1 to 3) refer to Section 3.1.4 .

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.7 BR=BYTE, BR!=BYTE

Branch to the instruction at the specified label if a specified byte in a 32-bit word matches or mismatches the `byte_compare_value`.

Instruction Format

<code>br=byte[reg, byte_no, byte_compare_value, label#], opt_tok</code>
<code>br!=byte[reg, byte_no, byte_compare_value, label#], opt_tok</code>

Parameter Descriptions

Parameter	Description
<code>reg</code>	Restricted source operand.
<code>byte_no</code>	A number specifying a byte in register to be compared with <code>byte_compare_value</code> . Valid <code>byte_no</code> values are 0 through 3. A value of 0 refers to the byte in bit position [7:0].
<code>byte_compare_value</code>	Value used for comparison. Valid <code>byte_compare_values</code> are 0 to 255.
<code>label#</code>	Symbolic label corresponding to the address of an instruction
<code>opt_tok</code>	<code>defer[n]</code> (n= 1 to 3) refer to Section 3.1.4 .

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.8 BR=CTX, BR!=CTX

Branch to the instruction at the specified label based on whether or not the current context is the specified context number.

Instruction Format

<code>br=ctx[ctx, label#], opt_tok</code>
<code>br!=ctx[ctx, label#], opt_tok</code>

Parameter Descriptions

Parameter	Description
<code>label#</code>	Symbolic label corresponding to the address of an instruction
<code>ctx</code>	Context number. Valid ctx values are: 4 context mode: 0, 2, 4, 6 8 context mode: 0 through 7.
<code>opt_tok</code>	<code>defer[n]</code> (n= 1 to 2) refer to Section 3.1.4 .

Condition Codes Affected

N	Z	V	C
Not Affected			



3.2.9 BR_INP_STATE, BR_!INP_STATE

Branch if the state of the specified state name is equal to 1 (BR_INP_STATE) or equal to 0 (BR_!INP_STATE). A state is information generated external to the Microengine, and sent as an input to the Microengine.

Instruction Format

<code>br_inp_state[state_name, label#], opt_tok</code>
<code>br_!inp_state[state_name, label#], opt_tok</code>

Parameter Descriptions

Parameter	State Number	State Name	Description	
state_name	0	NN_Empty	No valid data in NN_Ring from previous Neighbor ME.	
	1	NN_Full	NN_Ring in the Neighbor ME is full.	
	2	SCR_Ring0_Status	Rev A -- Indicates if the Scratch Ring is full (above the high water mark, which is a function of the Ring Size as defined in Scratch_Ring_Base_# CSR. Rev B -- Indicates either that the Scratch Ring is full (same as Rev A) or empty, based on the setting in Scratch_Ring_Base_#[Ring_Status_Flag].	
	3	SCR_Ring1_Status		
	4	SCR_Ring2_Status		
	5	SCR_Ring3_Status		
	6	SCR_Ring4_Status		
	7	SCR_Ring5_Status		
	8	SCR_Ring6_Status		
	9	SCR_Ring7_Status		
	10	SCR_Ring8_Status		
	11	SCR_Ring9_Status		
	12	SCR_Ring10_Status		
	13	SCR_Ring11_Status		
	14	FCI_Not_Empty.		FCI FIFO not empty.
	15	FCI_Full.		FCI FIFO full.
label#	A symbolic label corresponding to the address of an instruction			
opt_tok	defer[n] (n= 1 to 3) refer to Section 3.1.4 .			

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.10 BR_SIGNAL, BR_!SIGNAL

Branch if the specified signal (in the CTX_SIG_EVENTS register) is asserted or deasserted. If the signal is asserted, these instructions will clear it.

Instruction Format

<code>br_signal[signal_name, label#],opt_tok</code>
<code>br_!signal[signal_name, label#],opt_tok</code>

Parameter Descriptions

Parameter	Description
<code>label#</code>	Symbolic label corresponding to the address of an instruction.
<code>signal_name</code>	Symbolic label that specifies the signal. The two signals of a doubled signal is specified as <code>sig_name</code> and <code>sig_name</code> with a "+1" suffix (<code>sig_name+1</code>).
<code>opt_tok</code>	<code>defer[n]</code> (n= 1 to 3) refer to Section 3.1.4 .

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.11 BYTE_ALIGN_BE, BYTE_ALIGN_LE

This instruction requires multiple instructions and is used to concatenate data in two registers and put any four bytes into the destination register. The byte offset is specified by the value in the BYTE_INDEX local CSR. The BYTE_ALIGN_BE instruction supports big endian data and the BYTE_ALIGN_LE instruction supports little endian data.

One byte_align instruction is required to load the first four bytes from the src operand into one of two special temporary registers (not visible to the programmer). The BYTE_ALIGN_BE instruction uses the prev_B temporary register and the BYTE_ALIGN_LE uses the prev_A temporary register. For the IXP28x0 Rev B, Prev_A and Prev_B are loaded only during BYTE_ALIGN instructions.

Successive instructions use the src operand to specify the next 4 bytes, concatenate the 8 bytes, and select the desired result that is then placed into the dest register. The src operand is then placed into the appropriate temporary register to be used by the next byte_align instruction.

The number of cycles required to perform a byte alignment on “n” 4 byte words takes “n+2” cycles. (which include the instruction that loads the BYTE_INDEX local CSR and initialize the temp register).

Instruction Format

<code>byte_align_le [dest, src]</code>
<code>byte_align_be [dest, src]</code>

Parameter Descriptions

Parameter	Description
<code>dest</code>	Restricted destination that gets written with the result of the operation.
<code>src</code>	Restricted source operand.

Condition Codes Affected

N	Z	V	C
Result[31]==1	Result[31:0] == 0	cleared	cleared

Example 3-1. Big Endian Align - Byte Index = 2

The initial data is shown in Table 3-10 The NOP instructions are required to cover the read to use latency.

Table 3-10. Initial Register Contents

Register	Byte 3 [31:24]	Byte 2 [23:16]	Byte 1 [15:8]	Byte 0 [7:0]
0	0x00	0x01	0x02	0x03
1	0x04	0x05	0x06	0x07
2	0x08	0x09	0x0A	0x0B
3	0x0C	0x0D	0x0E	0x0F

Instruction	Prev B	A Operand	B Operand	Result (destx)
local_csr_wr[byte_index,2]				
nop				
nop				
nop				
Byte_align_be[--, r0]	--	--	0x00 01 02 03	--
Byte_align_be[dest1, r1]	0x00 01 02 03	0x00 01 02 03	0x04 05 06 07	0x02 03 04 05
Byte_align_be[dest2, r2]	0x04 05 06 07	0x04 05 06 07	0x08 09 0A 0B	0x06 07 08 09
Byte_align_be[dest3, r3]	0x08 09 0A 0B	0x08 09 0A 0B	0x0C 0D 0E 0F	0x0A 0B 0C 0D
NOTE: A Operand comes from Prev_B register during byte_align_be instructions.				

Example 3-2. Little Endian Align - Byte Index = 2

The initial data is shown in Table 3-11. The NOP instructions are required to cover the read to use latency.

Table 3-11. Initial Register Contents

Register	Byte 3 [31:24]	Byte 2 [23:16]	Byte 1 [15:8]	Byte 0 [7:0]
0	0x03	0x02	0x01	0x00
1	0x07	0x06	0x05	0x04
2	0x0B	0x0A	0x09	0x08
3	0x0F	0x0E	0x0D	0x0C

Instruction	A Operand	B Operand	Prev A	Result
local_csr_wr[byte_index,2]				
nop				
nop				
nop				
Byte_align_le[--, r0]	0x03 02 01 00	--		--
Byte_align_le[dest1, r1]	0x07 06 05 04	0x03 02 01 00	0x03 02 01 00	0x05 04 03 02
Byte_align_le[dest2, r2]	0x0B 0A 09 08	0x07 06 05 04	0x07 06 05 04	0x09 08 07 06
Byte_align_le[dest3, r3]	0x0F 0E 0D 0C	0x0B 0A 09 08	0x0B 0A 09 08	0x0D 0C 0B 0A
NOTE: B Operand comes from Prev_A register during byte_align_le instructions.				

Example 3-3. Big Endian Align using Index Address Mode to Local Memory

Another mode of operation is to use the index local CSRs with post-increment, to select the source and destination registers. The ACTIVE_LM_ADDR_0 local CSR is used to index into local memory and the T_INDEX_BYTE_INDEX local CSR is used to index the transfer registers and specify the byte alignment. The data in \$xfer0, to \$xfer3 is shown in Table 3-10. The NOP instructions are required to cover the read to use latency.

Instruction	Prev B	A Operand (Note 2)	B Operand	Result (LM)
alu[lm_index_byte_align, b_align, OR, lm_index0, <<2]	--	--	--	--
local_csr_wr[ACTIVE_LM_ADDR_0_BYTE_INDEX, lm_index_byte_align]	--	--	--	--
local_csr_wr[T_INDEX, &\$xfer0]	--	--	--	--
nop				
nop				
byte_align_be[--, *\$index++]	--	--	0x00 01 02 03	--
byte_align_be[*L\$index0[0], *\$index++]	0x00 01 02 03	0x00 01 02 03	0x04 05 06 07	0x02 03 04 05
byte_align_be[*L\$index0[1], *\$index++]	0x04 05 06 07	0x04 05 06 07	0x08 09 0A 0B	0x06 07 08 09
byte_align_be[*L\$index0[2], *\$index++]	0x08 09 0A 0B	0x08 09 0A 0B	0x0C 0D 0E 0F	0x0A 0B 0C 0D
NOTE: 1. b_align = byte align value (2 in this example) and \$xfer0 is starting register. Refer to T_INDEX, ACTIVE_LM_ADDR_0_BYTE_INDEX local CSR description. The & is a preprocessor directive that returns the address of the register. 2. A Operand comes from Prev_B register during byte_align_be instructions				

Example 3-4. Big Endian Align using Index Address Mode to GPR

A common function is to read a L2 and L3 header into transfer registers, determine the offset of the L3 header based on the L2 protocol field and then move the L3 header from the transfer registers to GPRs properly aligned. This is accomplished using the T_INDEX local CSR with post-increment. The T_INDEX_BYTE_INDEX local CSR is used to write both the T_INDEX and BYTE_INDEX local CSRs in a single write operation.

Instruction	Prev B	A Operand	B Operand	Result
alu[reg_addr, b_align, OR, &\$xfer0, <<2]	--	--	--	--
local_csr_wr[T_INDEX_BYTE_INDEX, reg_addr]	--	--	--	--
byte_align_be[--, *\$index++]	--	--	0x00 01 02 03	--
byte_align_be[gpr0, *\$index++]	0x00 01 02 03	0x00 01 02 03	0x04 05 06 07	0x02 03 04 05
byte_align_be[gpr1, *\$index++]	0x04 05 06 07	0x04 05 06 07	0x08 09 0A 0B	0x06 07 08 09
byte_align_be[gpr2, *\$index++]	0x08 09 0A 0B	0x08 09 0A 0B	0x0C 0D 0E 0F	0x0A 0B 0C 0D
NOTE: b_align = gpr containing byte align value (2 in this example) and \$xfer0 is starting register. The & is a preprocessor directive that returns the address of the register.				

3.2.12 CAM_CLEAR

Clears all entries in the MEv2 CAM by writing 0x00000000 to the tag, clearing all the state bits, and putting the LRU into an initial state where entry CAM 0 is LRU, ..., CAM entry 15 is MRU).

The CAM is not reset by ME reset. Software must either do a cam_clear prior to using the CAM to initialize the LRU and clear the tags to zero, or explicitly write all entries with cam_write.

Instruction Format

<code>cam_clear</code>

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.13 CAM_LOOKUP

Search the 16 entry MEv2 CAM for a 32-bit tag equal to the value specified in the `src_reg`. All entries are compared in parallel, and the result of the lookup is a 9 bit value which is written into the specified destination register in bits 11:3, with all other bits of the register zero. The result can also optionally be written into either of the `LM_Addr` registers.

The 9-bit result consists of 4 State bits (`dest_reg[11:8]`), concatenated with a 1-bit Hit/Miss indication (`dest_reg[7]`), concatenated with 4-bit entry number (`dest_reg[6:3]`). All other bits of `dest_reg` are written with 0. Possible results of the lookup are:

- miss (0)—lookup value is not in CAM, entry number is LRU (Least Recently Used) entry (which can be used as a suggested entry to replace), and State bits are 0000.
- hit (1)—lookup value is in CAM, entry number is entry which has matched, State bits are the value from the entry which has matched. The entry is marked as MRU (Most Recently Used).

An optional token allows the result to also be written into either of the `LM_Addr` registers.

The LRU (Least Recently Used) is maintained in a time-ordered CAM entry usage list. When an entry is loaded, or matches on a lookup, it is marked as MRU (Most Recently Used). Note that a lookup that misses does not modify the LRU list.

Note: The following rules must be followed to when using the CAM.

- CAM is not reset by ME reset. Software must either do a `CAM_clear` prior to using the CAM to initialize the LRU and clear the tags to zero, or explicitly write all entries with `CAM_write`.
- No two tags can be written to have same value. If this rule is violated, the result of a lookup that matches that value will be unpredictable, and LRU state is unpredictable.

The value 0x00000000 can be used as a valid lookup value. However, note that `CAM_clear` instruction puts 0x00000000 into all tags. So in order to not violate rule 2 after doing `CAM_clear`, it is necessary to write all entries to unique values prior to doing a lookup of 0x00000000.

Instruction Format

```
cam_lookup[dest_reg, src], opt_tok
```

Parameter Descriptions

Parameter	Description
<code>dest_reg</code>	Unrestricted destination that receives the result of the CAM lookup. Refer to Table 3-12 for result format.
<code>src</code>	Unrestricted source operand that holds the value to lookup in the CAM.
<code>opt_tok</code>	<code>lm_addr#[num]</code> : Load the result of the lookup into <code>LM_Addr</code> (<code># = 0 or 1</code>), as well as <code>dest</code> . Bits[6:3] of CAM result go to <code>LM_Addr[9:6]</code> . <code>Num</code> specifies a 2-bit value to load into <code>LM_Addr[11:10]</code> . <code>LM_Addr[5:0]</code> is set to zero. The write latency for loading the <code>LM_Addr</code> (<code># = 0 or 1</code>) is 3 cycles which is the same as if a <code>local_csr_wr</code> instruction were used.

Table 3-12. CAM_LOOKUP Result

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0		
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0				hit/miss	CAM Entry Number				0	0	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	state												

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.14 CAM_READ_TAG

Read the tag for the specified CAM entry into dest_reg.

Instruction Format

```
cam_read_tag[dest_reg, entry]
```

Parameter Descriptions

Parameter	Description
dest_reg	Unrestricted destination that receives the tag data from the CAM entry.
entry	Unrestricted source operand that specifies the CAM entry number to read. Valid values are 0 to 15.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.15 CAM_READ_STATE

Read the State bits for the specified CAM entry. The value is placed into bits [11:8] of `dest_reg`, with all other bits 0.

Instruction Format

```
cam_read_state[dest_reg, entry]
```

Parameter Descriptions

Parameter	Description
<code>dest_reg</code>	Unrestricted destination that receives the State bits from the CAM entry. <code>Dest_Reg</code> gets State in bits [11:8], all other bits are 0.
<code>entry</code>	Unrestricted source operand that specifies the CAM entry number to read. Valid values are 0 to 15.

Table 3-13. CAM_READ_STATE Result

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	state		0	0	0	0	0	0	0	0	0

Condition Codes Affected

N	Z	V	C
Not Affected			



3.2.16 CAM_WRITE

Write a 32-bit value in `src_reg` to the tag of the specified CAM entry. The entry is marked as MRU (Most Recently Used).

Note: No two tags can be written to have same value. If this rule is violated, the result of a lookup that matches that value will be unpredictable, and LRU state is unpredictable.

Instruction Format

<code>cam_write[entry_reg, src_reg, state_value]</code>

Parameter Descriptions

Parameter	Description
<code>entry</code>	Unrestricted source operand that specifies the CAM entry number to write. Valid values are 0 to 15.
<code>src_reg</code>	Unrestricted source operand that holds the data to write to the CAM entry.
<code>state_value</code>	Constant that specifies the State bit value.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.17 CAM_WRITE_STATE

Write the value into the State bits for the specified MEv2 CAM entry. The Tag value is not changed. This instruction does not modify the LRU list.

Instruction Format

<code>cam_write_state[entry, state_value]</code>
--

Parameter Descriptions

Parameter	Description
<code>entry</code>	Unrestricted source operand that specifies the CAM entry number to modify. Only the State bits are affected.
<code>state_value</code>	Constant that specifies the State bit value. Valid values are 0 to 0xF.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.18 CAP (Enumerated CSR Addressing)

Move data between the CAP CSRs and the ME. The CSR address is an enumerated constant that is specified at assembly time. Three commands are provided: read, write and fast_wr.

A fast_wr command eliminates the need to pull data from a transfer register during a write operation and therefore reduces the time required to complete the write operations. For read and write commands, xfer_data specifies an Transfer register. For fast_wr commands the xfer_data is either a 14-bit immediate (zero extended) or the keyword ALU. The keyword ALU specifies that the fast write data is 32-bits and is taken from the ALU output of the previous instruction.

Instruction Format

<code>cap[cmd, xfer_data, csr_addr], opt_tok</code>

Parameter Descriptions

Parameter	Cmd	Description	
cmd	read	Read the data from the CSR into an S or D Transfer register	
	write	Write the data to the CSR from an S Transfer register	
	fast_wr	Write the immediate data to the CSR	
xfer_data	read,	An S or D Transfer Read register Note 1	
	write	An S Transfer Write register. IXP28xx Rev B: An S or D Transfer Write register. Note 1	
	fast_wr	14-bit immediate (zero extended) data or keyword ALU. The keyword ALU specifies that the fast write data is 32-bits and is taken from the ALU output of the previous instruction.	
csr_addr	All cmds	The CAP CSR names listed in Table 3-14 defines the keywords that are used to address the CAP CSRs. Refer to the CSR definitions for a description of the CSR behavior.	
opt_tok	read, write	ctx_swap[sig_name] refer to Section 3.1.2.4 .	sig_done[sig_name] refer to Section 3.1.2.4 .
		indirect_ref refer to Section 3.1.2.3.1 .	defer[n] (n = 1 to 2) refer to Section 3.1.4 .
	fast_wr	indirect_ref (Only if xfer_data is not specified as ALU.)	ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3
1. In 4 context mode, only the S Transfer registers can be used			

Table 3-14. Enumerated CAP CSR Registers

Register Name	Comment
THD_MSG_SUMMARY_#_\$(# = {0,1}, \$ = {0,1})	Where: # = ME cluster number 0 to 1 \$ = register number 0 to 1
THD_MSG_#_\$_& (# = {0,1}, \$ = {0,7 or 3}, & = {0,7})	Where: # = ME cluster number 0 to 1 \$ = ME number in cluster 0 to 7 (IXP28x0) or 0 to 3 (IXP2400) & = thread number 0 to 7
THD_MSG (Generic)	
SELF_DESTRUCT_# (# = 0 - 1)	

Table 3-14. Enumerated CAP CSR Registers

Register Name	Comment
INTERTHREAD_SIG	
XSCALE_INT_# (# = A, B)	
SCRATCH_RING_BASE_# (# = 0 -15)	Can not be used with fast_wr command
SCRATCH_RING_HEAD_# (# = 0 - 15)	
SCRATCH_RING_TAIL_# (#= 0 - 15)	
HASH_MULTIPLIER_48_# (# = 0,1)	
HASH_MULTIPLIER_64_# (# = 0,1)	
HASH_MULTIPLIER_128_# (# = 0,1,2,3)	
PRODUCT_ID	
MISC_CONTROL	
IXP_RESET_0	
IXP_RESET_1	
CLOCK_CONTROL	
STRAP_OPTIONS	

Table 3-15. CAP Indirect Format (Read and Write Commands)

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
OV	ME		RES																Xfer Register Address				OV	OV	CTX						

Table 3-16. CAP Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
RES	Reserved
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127). Whether the register set is the S or D is specified in the instruction and cannot be changed using an indirect. Reserved for the fast write command.
OV [4]	Override bit for Xfer register Address field. Reserved for the fast write command.
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.19 CAP (Calculated Addressing)

Move data between the CAP devices and the ME. The CAP address is calculated at runtime by adding the `src_op1` and `src_op2` parameters. The entire CAP address space can be accessed including CAP CSRs, Timers, UART, PMU, SlowPort CSRs, SlowPort memory space, a remote ME's Local CSRs and Transfer Registers. When accessing a CSR, the `ref_cnt` value should be 1.

Note: This command is not supported in Rev A of the IXP2400.

Instruction Format

<code>cap[cmd, xfer, src_op1, src_op2, ref_cnt],opt_tok</code>
--

Parameter Descriptions

Parameter	Cmd	Description
<code>cmd</code>	read	Read the data from the CSR into an S or D Transfer register
	write	Write the data to the CSR from an S Transfer register
<code>xfer_data</code>	read,	An S or D Transfer Read register. Note 1
	write	An S Transfer Write register. IXP28xx Rev B: An S or D Transfer Write register. Note 1
<code>src_op1,</code> <code>src_op2</code>	All cmds	Restricted operands that define a byte address. The address is specified by <code>src_op1 + src_op2</code> . The CAP is accessed on 4 byte word boundaries so bits[1:0] are ignored by CAP.
<code>ref_cnt</code>	read, write	Reference count in increments of 4-byte words. Valid values are 1 to 8

Parameter Descriptions

Parameter	Cmd	Description	
opt_tok	read, write	ctx_swap[sig_name] refer to Section 3.1.2.4 .	sig_done[sig_name] refer to Section 3.1.2.4 .
		indirect_ref refer to Section 3.1.2.3.1 .	defer[n] (n = 1 to 2) refer to Section 3.1.4 .
		ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3	
		sig_remote[remote_sig_name, me_list]	Used when addressing another Microengines registers only. Signal the remote ME thread when the data has been pulled from or written to the remote thread's registers The me_list is a comma separated list that contains one or more ME numbers of the possible targets of the reflector operation. Valid values are 0 to 7 and 0x10 to 0x17. The remote signal must be manually allocated to the same address in all microengines. If either the ctx_arb[] or sig_done[] token has been specified, then the local signal must also be manually allocated to the same address. The linker will check the addresses for each microengine in the list.
1. In 4 context mode, only the S Transfer registers can be used			

[Table 3-17](#) illustrates the bit encodings for the 32-bit address derived from src_op1 + src_op2.

Table 3-17. CAP Bit Map Address Field Encoding (src_op1 + src_op2)

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
res		Not MBus (0)		res										xfer(0)	ME(1)	ME Cluster	ME number	S(0) D(1)	ME Xfer Registers					res					
res		Not MBus (0)		res										local csr(1)	ME(1)	ME Cluster	ME number	ME Local CSR address											
res		Not Mbus (0)		res						APB dev select Table 3-18		Not ME(0)	abp(0)	APB Address Table 3-18															
res		MBus (1)		res						res		ME(0)	csr(1)	CAP CSR Number															
res		MBus (1)		ROM Address (64M) The Slow Port controller supports two bus chip select asserted at the 32M boundary														res											

Note: S and D in bit [9] for ME transfer registers refer to the S and D transfer registers.

[Table 3-18](#) shows a simpler way of looking at the bit encodings for the 32-bit address derived from src_op1 + src_op2.

Table 3-18. CAP Calculated Address Field Encoding (src_op1 + src_op2)

CAP Addressable Devices	Address Range	Comments
GPIO	0x0001 0000 CSR address	Refer to Section 5.6.6 . APB dev select = 0
CAP CSRs	0xC000 4800 CSR address	Refer to Section 5.6.1 -Scratchpad CSRs
	0xC000 4900 CSR address	Refer to Section 5.6.2 - Hash CSRs
	0xC000 4000 CSR address	Refer to Section 5.6.3 - Fast Write CSRs
	0xC000 4A00 CSR address	Refer to Section 5.6.4 - Global CSRs
ME Transfer Registers (Reflector)	0x0000 8000 ME Cluster ME Number Xfer address	Remote MEs Transfer Registers Where: [14]= ME cluster [13:10]= ME number [9:2]= Xfer Register address [1:0]= ignored
ME Local CSRs (Reflector)	0x0001 8000 ME Cluster ME Number CSR address	Where: [14]= ME cluster [13:10]= ME number [9:0]= CSR address
Timers	0x0002 0000 CSR address	Refer to Section 5.6.5 . APB dev select = 2
UART	0x0003 0000 CSR address	Refer to Section 5.6.7 . APB dev select = 3
PMU	0x0005 0000 CSR address	Refer to Section 5.6.8 . APB dev select = 5
SlowPort CSRs	0x0008 0000 CSR address	Refer to Section 5.6.9 . APB dev select = 8
SlowPort (Memory Space)	0x0400 0000–0x07ff ffff	64M address range. The Slow Port controller generates a two Slow port bus chip select at the 32M boundary.
Note: “ ” is the bit wise OR operator		

Table 3-19. CAP Indirect Format (Read and Write Commands)

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
OV	ME					OV	Ref_Cnt				RES				Xfer Register Address				OV	OV	CTX										

Table 3-20. CAP Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
OV [25]	Override bit for Ref_Cnt field

Table 3-20. CAP Field Definitions

Field	Description
Ref_Cnt	Overrides the ref_cnt field specified by the instruction. This field supports ref_cnts from 0 to 15 where a value of 0 indicates a transfer of 1 and 15 a transfer of 16. Note that when specifying the Ref_cnt and Xfer Register Address, data can be written into another contexts registers set. The max_nn optional token should be used with this token. Refer to Section 3.1.2.3.2 .
RES	Reserved
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127). Whether the register set is the S or D is specified in the instruction and cannot be changed using an indirect.
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.20 CAP (Reflect)

Move data between an MEs Transfer registers and another MEs Transfer registers. The Transfer register accepting the data can be either an S or a D. The Transfer register sourcing the data must be an S. In the IXP28xx rev B, the transfer register sourcing the data can be either an S or a D.

Note this version of the CAP instruction used to be the reflect instruction in earlier versions of the assembler.

Instruction Format

cap[cmd, xfer, rem_ME, rem_reg, rem_ctx, ref_cnt],opt_tok

Parameter Descriptions

Parameter	Cmd	Description	
cmd	read	Read data from the remote MEs write transfer register into this MEs read transfer register.	
	write	Write data from this MEs write transfer register to the remote MEs read transfer register.	
xfer	read	An S or D Transfer Read register in this ME used to hold the data that is read from the remote ME. Note 1	
	write	An S-Transfer Write register in this ME used to hold the data that is written to the remote ME. IXP28xx Rev B: An S or D Transfer Write register. Note 1	
rem_ME	All cmds	Number of remote ME. Valid values are 0 to 7 and 0x10 to 0x17	
rem_reg	All cmds	A Transfer register in the remote ME. For Read, the remote register must be an S-Transfer, For Write the remote register can be either an S or a D register. IXP28xx Rev B: For read, the remote register can be either an S or D Transfer Write register.	
rem_ctx	All cmds	Context number of the remote transfer register. Valid values are 0 to 7. (Note 2)	
ref_cnt	All cmds	Reference count in increments of 4 byte words. Valid values are 1 to 8.	
opt_tok	All cmds	ctx_swap[sig_name]	sig_done[sig_name]
		refer to Section 3.1.2.4 .	refer to Section 3.1.2.4 .
		indirect_ref	defer[n] (n = 1 to 2) refer to
		refer to Section 3.1.2.3.1 .	Section 3.1.4 .
		sig_remote[remote_sig_name]	ind_targets[me1, me2, ...]
		(Note 3)	refer to Section 3.1.2.3.3
<ol style="list-style-type: none"> 1. In 4 context mode, only the S Transfer registers can be used 2. If the remote ME is in four-context mode, then the context should be given as 0, 2, 4, or 6. 3. Signal the remote ME thread when the data has been pulled from or written to the remote thread's registers. The remote ME is given by the instruction parameter. If either the ctx_arb[] or sig_done[] token has been specified, the local and remote signals must be manually allocated to the same address. The linker will verify this. 			

Table 3-21. CAP (Reflect) Indirect Format

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
OV	ME				OV	Ref_Cnt				RES				Xfer Register Address				OV	OV	CTX										

Table 3-22. CAP (Reflect) Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	For a read operation, it indicates the ME where the destination register resided. For a write operations, it indicates the ME where the source register resides. It also specifies the ME that is signaled when the sig_done (and sig_remote) optional tokens is specified. Refer to Section 3.1.2.3.3 .
OV [25]	Override bit for Ref_Cnt field
Ref_Cnt	Overrides the ref_cnt field specified by the instruction. This field supports ref_cnts from 0 to 15 where a value of 0 indicates a transfer of 1 and 15 a transfer of 16. Note that when specifying the Ref_cnt and Xfer Register Address, data can be written into another contexts registers set. If this field is changed, the ref_cnt field in the instruction should be a keyword max_nn (where nn = 1-16). Refer to Section 3.1.2.3.2 .
RES	Reserved
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127). Whether the register set is the S or D is specified in the instruction and cannot be changed using an indirect.
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.21 CRC_LE, CRC_BE

Enable the CRC datapath to compute a CRC. The CRC is calculated using a remainder that resides in the CRC_Remainder Local CSR and the source data. The result is written to the CRC_Remainder Local CSR and the unmodified source data can be written to a destination register. This allows a CRC to be calculated and the result moved from one register to another (example: S-transfer in to S-transfer out) in the same instruction.

The CRC_Remainder Local CSR is typically initialized prior to doing the CRC instruction(s). CRC instruction can not be used consecutively. At least one intervening instruction must be done between two CRC instructions, and 5 intervening instruction before reading the CRC result.

[Example 3-1](#) shows how a CRC can be calculated over a block of data by interleaving the CRC instructions.

The `crc_le` instruction (little endian) is used when the data is in little endian mode so the bytes are swapped before the CRC is performed. The `crc_be` instruction (big endian) is used when the data is in big endian format and no bytes are swapped before the CRC is performed. Refer to [Example 3-1](#) to [Example 3-4](#).

Instruction Format

```

crc_le[crc_type, dest_reg, src], opt_tok
crc_be[crc_type, dest_reg, src], opt_tok
    
```

Parameter Descriptions

Parameter		Description
crc_type#	<code>crc_ccitt</code>	CRC-CCITT polynomial is: $x^{16} + x^{12} + x^5 + 1$
	<code>crc_32</code>	CRC-32 polynomial is: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$.
	<code>crc_iscsi</code>	IXP28xx Rev B only. CRC-iscsi polynomial is: $x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^6 + 1$.
	<code>crc_10</code>	IXP28xx Rev B only. CRC-10 polynomial is: $x^{10} + x^9 + x^5 + x^4 + x + 1$.
	<code>crc_5</code>	IXP28xx Rev B only. CRC-5 polynomial is: $x^5 + x^2 + 1$.
	<code>none</code>	Moves the data from the source to the destination without performing any CRC operation on it.
dest	All <code>crc</code> types	Unrestricted destination that gets written with the unmodified <code>src</code> operand.
src	All <code>crc</code> types	Unrestricted source operand.

Parameter Descriptions

Parameter	Description			
opt_tok	crc_ccitt crc_32 crc_iscsi crc_10 crc_5	One of the following tokens can be used to limit which bytes of data are used in the computation. If no token is used, all four bytes are used. These are used in the beginning and end of a CRC computation to skip leading and trailing byte positions. The meaning of the token can be thought of in two ways: 1. It specifies the bytes, in order of left to right, of the post-swapped big endian data. 2. It specifies the bytes of the pre-swapped data in the order implied by the endianness		
		opt_tok	Bytes in source used for CRC calculation	
			Big endian	Little Endian
		bytes_0_3	0,1,2,3	3,2,1,0
		bytes_0_2	0,1,2,-	-,2,1,0
		bytes_0_1	0,1,-,-	-,-,1,0
		byte_0	0,-,-,-	-,-,-,0
		bytes_1_3	-,1,2,3	3,2,1,-
		bytes_2_3	-,-,2,3	3,2,-,-
		byte_3	-,-,-,3	3,-,-,-
All crc types	bit_swap: Swaps the bits in each individual byte so that bit 7 is swapped with bit 0, bit 6 is swapped with bit 1, bit 5 is swapped with bit 2, and bit 4 is swapped with bit 3.			

Condition Codes Affected

N	Z	V	C
Result[31] == 1	Result[31:0] == 0	Cleared	Cleared

Example 3-1. Loading, Calculating and Reading the CRC

```

; Assume data to CRC is in $s_transfer_in[3:0]
; Current remainder has been put into gpr_n
; Compute CRC while moving data to d_transfer_out[3:0]
local_csr_wr[crc_remainder, gpr_n] ;restore remainder
nop ; nops could be replaced by useful instructions
nop
nop
crc[crc_ccitt, $$d_trans_out_0, $s_trans_in_0]
nop
crc[crc_ccitt, $$d_trans_out_1, $s_trans_in_1]
nop
crc[crc_ccitt, $$d_trans_out_2, $s_trans_in_2]
nop
crc[crc_ccitt, $$d_trans_out_3, $s_trans_in_3]
nop
nop
nop
nop
nop ; five intervening instructions before reading result
local_csr_rd[crc_remainder] ;get the new remainder
immed[gpr_n, 0x0000]
    
```

Example 3-2. crc_le[] with bytes_0_1

```

case 1#:
;data from alu out:  M L C S

    crc_le[], bytes_0_1

;data after swapping: S C L M
;CRC computed on:    S C

```

Example 3-3. crc_be[] with bytes_2_3

```

case 2#:
;data from alu out:  M L C S

    crc_be[], bytes_2_3

;data after swapping: M L C S
;CRC computed on:    C S

```

Example 3-4. crc_le[] with bytes_2_3

```

case 3*:
;data from alu out:  M L C S

    crc_le[], bytes_2_3

;data after swapping: S C L M
;CRC computed on:    L M

```

Example 3-5. crc_be[] with bytes_0_1

```

case 4#:
;data from alu out:  M L C S

    crc_be[], bytes_0_1

;data after swapping: M L C S
;CRC computed on:    M L

```

3.2.22 CTX_ARB

Swap the currently running context out to let another context execute. Wake up the swapped out context when the specified signal(s) is activated.

Instruction Format

```
ctx_arb[Event_Signal_Mask], op_tok
```

Parameter Descriptions

Parameter	Mask option	Description
Event_Signal_Mask	signal list	A list of signal names separated by commas. The thread is put to sleep and woken based on the state of the signals and the ANY and ALL optional tokens.
	Voluntary	Voluntary is a keyword that indicates that the thread should be put to sleep and woken when all the other threads have had a change to run.
	Kill	Kill is a keyword that indicates the current thread should be put to sleep and never woken.
	bpt	bpt is a keyword that indicates the current thread should be put to sleep, no thread should be woken, and the Intel XScale® core should be interrupted. This is typically used for breakpoints. The actions taken are as follows: -Clear the CTX_Enable for all contexts in the ME. -Put the currently executing Context into Sleep state. -Sets the CTX_Enable[Breakpoint] bit in the ME local CSR, which will cause the attn bit to assert in the (IRQ,FIQ)ATTN_STATUS Intel XScale® core Local CSR. Note that the CTX_WAKEUP_EVENTS is cleared for the context that is running so the Intel XScale® or PCI Host software must set the voluntary bit to restart the context.
	--	The value "--" indicates that the instruction does not contain a list of Event Signals; that list must instead be loaded using the instruction local_csr_wr[CTX_Wakeup_Events, src]. This can be the instruction that immediately precedes the ctx_arb[--] instruction, or in the defer shadows of the ctx_arb instruction. The local_CSR_wr can be to either CTX_Wakeup_Events_Active or CTX_Wakeup_Events_Indirect CSRs.
opt_tok	All mask options	defer[n] (n = 1 to 2)
	signal list, --	ALL: Activate the context when all of the listed signal(s) is received. This also clears the contexts sig_events and wakeup_events when the context is woken. If no token is used, ALL is assumed. ALL and ANY are mutually exclusive. ANY: Activate the context when any of the listed signal(s) is received. This also clears the context wakeup_events when the context is put in Ready state, but leave the contexts sig_events unchanged. The thread can then use the br_signal or br_!signal to determine which signal event woke it. ANY and ALL are mutually exclusive.
	signal list, --, Voluntary	br[label#] - Resume execution at the address specified by the label when the context wake up. If this token is not used executions resumes at the next sequential instruction. The defer optional token can be used with this token.

Example 3-1. CTX_ARB

```
;example 1: signal list mask option with optional token = any
ctx_arb[test1, test2], any

;example 2: -- mask option
ctx_arb[ --],defer[1]
local_csr_wr[ACTIVE_CTX_WAKEUP_EVENTS, 0x2] ; wake on signal 1

;example 3: branch optional token with --
ctx_arb[--],br[label#],defer[1]
local_csr_wr[ACTIVE_CTX_WAKEUP_EVENTS, 0x2] ; wake on signal 1

;example 4: Voluntary mask option
ctx_arb[voluntary]
```

3.2.23 DBL_SHF

Load a destination register with a 32-bit word that is formed by concatenating the A operands and B operands together (A is most significant, B is least significant), right shifting the 64-bit quantity by the specified amount, and then storing the lower 32 bits.

Instruction Format

```
dbl_shf[dest, A_op, B_op, shf_cntl]
```

Parameter Descriptions

Parameter	Description
dest	Restricted destination that gets written with the result of the operation.
A_op, B_op	Restricted source operand.
shf_cntl	>>1 through >>31: (Right shift 1) through (Right shift 31)
	>>indirect : Right shift by the indirect value. The indirect value is specified by the previous instruction which must be an ALU or ALU_SHF and the shift amount is specified in the lower 5 bits of the A_op parameter (which must be a register - not a constant).

Condition Codes Affected

N	Z	V	C
Result[31]==1	Result[31:0] == 0	cleared	cleared

Example 3-1. DBL_SHF >>value

```
If a = 0x87654321 and b = 0xFEDCBA98, then

dbl_shf[c, a, b, >>12]

saves 0x321FEDCB in c.
```

Example 3-2. DBL_SHF >>indirect

```
If a = 0x87654321, b = 0xFEDCBA98, shf_value = 12, any_reg = any value (it's not
used) then

alu[--, shf_value, OR, any_reg]
dbl_shf[c, a, b, >>indirect]

saves 0x321FEDCB in c.
```

3.2.24 DRAM (Read and Write)

Move data between DRAM and the ME Transfer registers.

Instruction Format

```
dram[cmd, xfer, src_op1, src_op2, ref_cnt],opt_tok
```

Parameter Descriptions

Parameter	Cmd	Description	
cmd	read	Read from DRAM to S or D Transfer registers.	
	write	Write to DRAM from D Transfer registers.	
xfer	read	A S or D Transfer Read register. Note 1	
	write	A D-Transfer Write register. IXP28xx Rev B: An S or D Transfer Write register. Note 1	
src_op1, src_op2	All cmds	Restricted source operands that define the DRAM byte address. The address is specified by src_op1 + src_op2. The DRAM address begins at 0 and ends at the max memory installed in the system. DRAM is always accessed on 8-byte word boundaries and therefore the low three bits of the byte address are ignored by the DRAM channel.	
ref_cnt	All cmds	Reference count in 8-byte words. Valid values are 1-8.	
opt_tok	All cmds	sig_done[sig_name2] refer to Section 3.1.2.4 .	ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3
		indirect_ref refer to Section 3.1.2.3.1 .	
	read	ignore_data_error	

1. In 4 context mode, only the D Transfer registers can be used

Table 3-23. DRAM Indirect Format

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
OV	ME				OV	Ref Cnt				OV	Byte Mask				Xfer Register Address				OV	OV	CTX										

Table 3-24. DRAM Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
OV [25]	Override bit for Ref_Cnt field.
Ref_Cnt	Overrides the ref_cnt field specified by the instruction. This field supports ref_cnts from 0 to 15 where a value of 0 indicates a transfer of 1 and 15 a transfer of 16. Note that when specifying the Ref_cnt and Xfer Register Address, data can be written into another contexts registers set. If this field is changed, the ref_cnt field in the instruction should be a keyword max_nn (where nn = 1-16). Refer to Section 3.1.2.3.2 .
OV [20]	Override bit for Byte Mask field

Table 3-24. DRAM Field Definitions

Field	Description
Byte Mask	This field should only be used when ref_cnt is set to a transfer of 1. It selects which bytes will be written to DRAM memory during a write operation. A “1” signifies the byte will be written where bit 12 corresponds to bits 7:0 and bit 19 corresponds to bits 63:56. Note that this may result in a read modify write operation that is performed at the DRAM channel.
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127). Whether the register set is the S or D is specified in the instruction and cannot be changed using an indirect.
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

Table 3-26. DRAM RBUF_RD & TBUF_WR Field Definitions

Field	Description
RBUF/TBUF Byte Address	This is required to specify the RBUF and TBUF address. The other fields are optional (except the override bit for this field). The TBUF and RBUF are mapped to the same 8K addresses. TBUF is accessed by a write while the RBUF is accessed by a read. The 8k address range specified in the indirect begins at 0x2000 and ends at 0x3FFF. The TBUF and RBUF can only be read on 8 byte word boundaries and therefore the low three bits of the address are ignored.
OV [4]	Override bit for RBUF/TBUF Byte Address field and must always be set
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.26 FFS

Find First bit set in the src register beginning at the LSB. Dest[4:0] gets the encoded value from 0 to 31 indicating the least significant '1' bit, in the src. If there are no bits set, the Z condition code is set; otherwise the Z condition code is cleared.

Instruction Format

<code>ffs [dest, src]</code>

Parameter Descriptions

Parameter	Description
<code>dest</code>	Unrestricted destination that receives the result of the operation. Dest[4:0] receives a value from 0 to 31 indicating first bit found starting at the LSB in the src. If there are no bits set, the Z condition code is set; otherwise the Z condition code is cleared. All other bits of dest receive '0'.
<code>src</code>	Unrestricted operand for the operation (B operand).

Condition Codes Affected

N	Z	V	C
Cleared	Set if B == 0	Cleared	Cleared

3.2.27 HALT

This instruction puts the current thread to sleep without waking up any other thread and interrupts the Intel XScale® core. This instruction is equivalent to instruction "ctx_arb[bpt]". The actions taken are as follows:

- Clear the CTX_Enable for all contexts in the ME.
- Put the currently executing Context into Sleep state.
- Sets the CTX_Enable[Breakpoint] bit in the ME local CSR, which will cause the attn bit to assert in the {IRQ,FIQ}ATTN_STATUS in the Intel XScale® core Local CSR.

Note that the CTX_WAKEUP_EVENTS is cleared for the context that is running so the Intel XScale® core or PCI Host software must set the voluntary bit to restart

Instruction Format

HALT

3.2.28 HASH

Issue a command to the Hash Unit requesting that the data in the transfer registers be moved to the hash unit, a hash operation performed on the data, and the result returned back to the transfer registers. Three variations of the instruction allow the hash data to be 48, 64 or 128 bits long. The ref count specifies the number of hash operations to be perform. Data should be placed into the transfer registers as shown in [Table 3-30](#). The Hash unit uses a multiplier to generate the hash data. The multiplier is set via the CAP CSRs HASH_MULTIPLIER_\$\$_# (where \$\$ = 48, 64 or 128, and # is the register number).

Instruction Format

```
hash_48 [xfer, ref_cnt],opt_tok
hash_64 [xfer, ref_cnt],opt_tok
hash_128 [xfer, ref_cnt],opt_tok
```

Parameter Descriptions

Parameter	Instruction	Description	
Instruction	hash_48	Perform 1 to 3 hash operations on 48 bits of data	
	hash_64	Perform 1 to 3 hash operations on 64 bits of data	
	hash_128	Perform 1 to 3 hash operation on 128 bits of data	
xfer	All instructions	Specifies both are read and write S-Transfer register. The write S-Transfer register is used to hold the data to be hashed while the result is placed into the read S-Transfer register. Table 3-27 shows the number of S-Transfer registers used for each instruction for each ref_cnt IXP28xx Rev B: An S or D Transfer Write register can be used Note 1	
ref_cnt	All instructions	Specifies the number of hash operations to perform. Valid values are 1-3.	
opt_tok	All instructions	sig_done[sig_name2] refer to Section 3.1.2.4 .	ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3
		indirect_ref refer to Section 3.1.2.3.1 .	
1. In 4 context mode, only the S Transfer registers can be used			

Table 3-27. Number of S-Transfer Registers Used by Hash Instruction

Instruction	ref_cnt		
	1	2	3
Hash_48	2 read 2 write	4 read 4write	6 read 6write
Hash_64	2 read 2 write	4 read 4write	6 read 6 write
Hash_128	4 read 4write	8 read 8write	12 read 12 write

Table 3-28. Hash Indirect Format

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
OV	ME				OV	RES	ref_cnt	RES											Xfer Register Address				OV	OV	CTX						

Table 3-29. Hash Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
OV [25]	Override bit for Ref_Cnt field
REF_CNT	Overrides the ref_cnt field specified by the instruction. Valid values are: 1 = one hash operation 2 = two hash operation, 3 = three hash operation
RES	Reserved
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the S transfer registers (0-127).
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

Table 3-30. Data Format in Transfer Registers

31	0	transfer register	31	0	transfer register
don't care	hash 3[47:32]	\$xfer n+5	hash 3[127:96]	\$xfer n+11	
	hash 3[31:0]	\$xfer n+4	hash 3[95:64]	\$xfer n+10	
don't care	hash 2[47:32]	\$xfer n+3	hash 3[63:32]	\$xfer n+9	
	hash 2[31:0]	\$xfer n+2	hash 3[31:0]	\$xfer n+8	
don't care	hash 1[47:32]	\$xfer n+1	hash 2[127:96]	\$xfer n+7	
	hash 1 [31:0]	\$xfer 0	hash 2[95:64]	\$xfer n+6	
48 bit Hash			hash 2[63:32]	\$xfer n+5	
			hash 2[31:0]	\$xfer n+4	
31	0		hash 1[127:96]	\$xfer n+3	
	hash 3[63:32]	\$xfer n+5	hash 1[95:64]	\$xfer n+2	
	hash 3[31:0]	\$xfer n+4	hash 1[63:32]	\$xfer n+1	
	hash 2[63:32]	\$xfer n+3	hash 1[31:0]	\$xfer 0	
	hash 2[31:0]	\$xfer n+2			

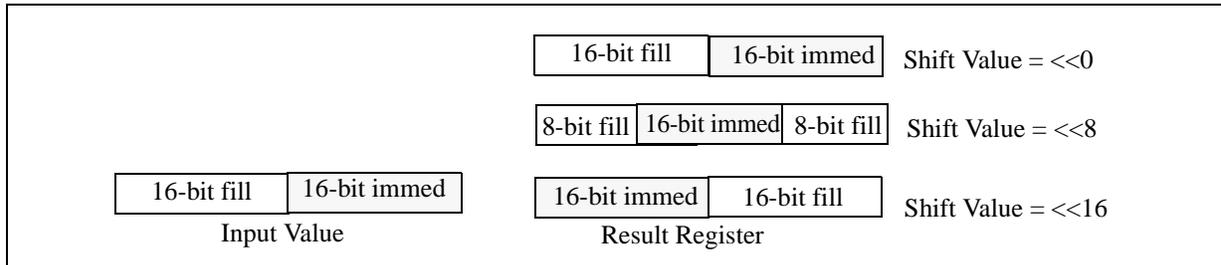
Table 3-30. Data Format in Transfer Registers



3.2.29 IMMED

Load immediate 16-bits into the specified register. The immediate data must be specified having the upper 16-bits equal to either all zeros or ones. As in Figure 3-1, the immediate data can be stored in the 32-bit word aligned on an 8-bit boundary based on the optional shift parameter. The fill data is either all zeros or ones and is based on the specified upper 16-bits.

Figure 3-1. Load Immediate



Instruction Format

```
immed[dest, immed_data, shf_cntl]
```

Parameter Descriptions

Parameter	Description
dest	Unrestricted operand. If the register is specified as a transfer register the result is always placed into the transfer out register.
immed_data	32-bit data having the upper bits all zeroes or all ones and the lower 16-bits user defined.
shf_cntl	0, <<0, or, no character : No shift.
	<<8 : Left shifts one byte.
	<<16: Left shifts two bytes.

Condition Codes Affected

N	Z	V	C
Not Affected			

Example 3-1. IMMED

```
;Negative number (-25610 = 0xFFFF FF00)
immed[dest_reg10, -256, <<0];Result: 0xffff ff00
immed[dest_reg11, -256, <<8];Result: 0xffff 00ff
immed[dest_reg12, -256, <<16];Result: 0xff00 ffff
```

Example 3-2. IMMED

```
;Hexadecimal number - zero fill
immed[dest_reg04, 0xff00, <<0] ;Result: 0x0000 ff00
immed[dest_reg05, 0xff00, <<8] ;Result: 0x00ff 0000
immed[dest_reg06, 0xff00, <<16] ;Result: 0xff00 0000
```

Example 3-3. IMMED

```
;Hexadecimal number - one fill  
immed[dest_reg07, 0xffffffff00, <<0];Result 0xffff ff00  
immed[dest_reg08, 0xffffffff00, <<8];Result 0xffff 00ff  
immed[dest_reg09, 0xffffffff00, <<16];Result 0xff00 ffff
```

Example 3-4. IMMED

```
;Negative number (-25610 = 0xFFFF FF00)  
immed[dest_reg10, -256, <<0];Result: 0xffff ff00  
immed[dest_reg11, -256, <<8];Result: 0xffff 00ff  
immed[dest_reg12, -256, <<16];Result: 0xff00 ffff
```

3.2.30 IMMED_B0, IMMED_B1, IMMED_B2, IMMED_B3

The specified `dest_reg` is read as a source, one byte of immediate data is loaded into the specified byte, and the result is written into the destination, while preserving all the other bits of the source value. These instructions perform a read-modify-write operation on a specified destination register.

If a Transfer register is specified as the `dest_reg`, these instructions perform a read and modify from a read transfer register and write the result into a transfer out register.

If a Neighbor register is specified, the Microengines Next Neighbor register is read, modified and then stored into the Next Neighbor Microengine.

B0 refers to the least significant byte.

Instruction Format

<code>immed_b0[dest_reg, byte_data]</code>
<code>immed_b1[dest_reg, byte_data]</code>
<code>immed_b2[dest_reg, byte_data]</code>
<code>immed_b3[dest_reg, byte_data]</code>

Parameter Descriptions

Parameter	Description
<code>dest_reg</code>	Unrestricted operand that receives the result of the operation.
<code>immed_data</code>	Immediate byte to be loaded into <code>dest_reg</code> .

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.31 IMMED_W0, IMMED_W1

The specified `dest_reg` is read as a source, one word of immediate data is loaded into the specified word, and the result is written into the destination, while preserving all the other bits of the source value. These instructions perform a read-modify-write operation on a specified destination register.

If a Transfer register is specified as the `dest_reg`, these instructions perform a read and modify from a read transfer register and writes the result into a write transfer register.

If a Neighbor register is specified, the Microengine's Next Neighbor register is read, modified and then stored into the Next Neighbor Microengine.

W0 refers to the least significant word. W1 refers to the most significant word.

Instruction Format

<code>immed_w0[dest_reg, immed_data]</code>
<code>immed_w1[dest_reg, immed_data]</code>

Parameter Descriptions

Parameter	Description
<code>dest_reg</code>	Unrestricted operand that receives the result of the operation.
<code>immed_data</code>	Immediate 2 bytes to be loaded into <code>dest_reg</code> . Valid <code>immed_data</code> values are 0 to 0xFFFF

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.32 JUMP

Unconditional branch to an address that is formed during runtime execution by the addition of the register and label# values.

Instruction Format

`jump[src, label#], opt_tok`

Parameter Descriptions

Parameter	Description
src	Unrestricted operand that holds the offset into the jump table. Immediate data is not supported.
label#	Symbolic label corresponding to the base address of a jump table.
opt_tok	defer[n] (n= 1 to 3) refer to Section 3.1.4 . targets [label1, label2, ...labeln] This is always required. It is a list of labels that represents all possible locations to which the jump could occur. All possible targets must be specified in the target list, otherwise the code produced by the assembler may not run as expected.

Condition Codes Affected

N	Z	V	C
Not Affected			

Example 3-1. Jump

```

;Possible Results:
;if offset = 0, result = 1
;if offset = 2, result = 2
;if offset = 4, result = 3
    immed[offset, 2]
    jump[offset, base0#], targets [base0#, base1#, base2#]
continue#:
    br[continue#]
base0#;base0 + 0
    immed[result, 1]
    br[continue#]
base1#;base0 + 2
    immed[result, 2]
    br[continue#]
base2#;base0 + 4
    immed[result, 3]
    br[continue#]
    
```

3.2.33 LD_FIELD, LD_FIELD_W_CLR

Load one or more bytes positions within a register with the shifted value of another operand. Data in the bytes that are not loaded remain unchanged or are cleared. LD_FIELD performs a read-modify-write on a destination register. LD_FIELD_W_CLR performs a write to a destination register.

If a Transfer register is specified as the dest_reg for LD_FIELD, these instructions perform a read and modify from a read transfer register and writes the result into a write transfer register.

If a Neighbor register is specified for LD_FIELD, the Microengine's Next Neighbor register is read, modified and then stored into the Next Neighbor Microengine.

Instruction Format

<code>ld_field[dest_reg, byte_enables, src_op, opt_shf_cntl], op_tok</code>
<code>ld_field_w_clr[dest_reg, byte_enables, src_op, opt_shf_cntl], op_tok</code>

Parameter Descriptions

Parameter	Description
<code>dest_reg</code>	Restricted operand that receives the result of the operation. Note that this operand is also used as a source. Note that *n\$index++ can not be used as destination for ld_field_w_clr instruction.
<code>byte_ld_enables</code>	A 4-bit mask that specifies which byte(s) are affected by the instruction. Each set bit enables the corresponding byte of the destination operand 32-bit word to be loaded or cleared. There must be at least 1 set bit in this mask. For example, 0101 loads the 1st and 3rd bytes while the other bytes remain unchanged.
<code>src_op</code>	Restricted source. If a GPR, this register must be on the opposite bank as the destination register. Refer to Table 3-3 for source register selection rules.
<code>opt_shf_cntl</code>	Shift or rotate the source_op contents using the syntax shown below.
	<<n: Left shift n bits, where n = 1 through 31.
	<<indirect: Left shift by an amount specified in the lower 5 bits of the A operand of the previous instruction (the previous instruction must be one of the following ALU or ALU_SHF instructions -- A AND B, A AND ~B, A XOR B, A OR B). The lower 5 bits of the A operand should be n, where n is the desired left shift amount.
	>>n: Right shift n bits, where n = 1 through 31.
	>>indirect: Right shift by the amount specified in the lower 5 bits of the A operand of the previous instruction.
	<<rotn: Left rotate n bits, where <<rot is a keyword and n = 1 to 31.
>>rotn : Right rotate n bits, where >>rot is a keyword and n = 1 to 31.	
<code>opt_tok</code>	load_cc :: Load ALU condition codes based on result formed

Condition Codes Affected

Optional token	N	Z	V	C
load_cc	Result[31] == 1	Result[31:0] == 0	Cleared	Cleared
not load_cc	Not Affected			

3.2.34 LOAD_ADDR

Load a register with an address of the location specified by label#.

Instruction Format

<code>load_addr [dest, label#]</code>

Parameter Descriptions

Parameter	Description
<code>dest</code>	Restricted destination that receives the result of the operation.
<code>label#</code>	Symbolic label corresponding to the address of an instruction.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.35 LOCAL_CSR_RD

Read the specified Microengine CSR register. The read data is accessed by replacing the immediate data source operand of the next immed instruction with the Microengine CSR read data. If the very next instruction does not contain an immediate data source operand field, then the opportunity to access the CSR data read from the previous instruction is lost.

Instruction Format

```
local_csr_rd[local_csr]
```

Parameter Descriptions

Parameter	Description	
local_csr	Specifies the Local CSR. See Section 5.2, "Microengine Local CSRs" for a description of the CSRs.	
	CSR Names	
	USTORE_ADDRESS	INDIRECT_LM_ADDR_1
	ALU_OUT	INDIRECT_LM_ADDR_1_BYTE_INDEX
	CTX_ARB_CNTL	ACTIVE_LM_ADDR_1
	CTX_ENABLES	ACTIVE_LM_ADDR_1_BYTE_INDEX
	CC_ENABLE	BYTE_INDEX
	CSR_CTX_POINTER	T_INDEX
	INDIRECT_CTX_STS	T_INDEX_BYTE_INDEX
	ACTIVE_CTX_STS	INDIRECT_FUTURE_COUNT_SIGNAL
	ACTIVE_CTX_SIG_EVENTS	ACTIVE_FUTURE_COUNT_SIGNAL
	INDIRECT_CTX_SIG_EVENTS	NN_PUT
	INDIRECT_CTX_WAKEUP_EVENTS	NN_GET
	ACTIVE_CTX_WAKEUP_EVENTS	TIMESTAMP_HIGH, TIMESTAMP_LOW
	INDIRECT_CTX_FUTURE_COUNT	CRC_REMAINDER
	ACTIVE_CTX_FUTURE_COUNT	PROFILE_COUNT
	INDIRECT_LM_ADDR_0	PSEUDO_RANDOM_NUMBER
	INDIRECT_LM_ADDR_0_BYTE_INDEX	LOCAL_CSR_STATUS
	ACTIVE_LM_ADDR_0	
ACTIVE_LM_ADDR_0_BYTE_INDEX		

Condition Codes Affected

N	Z	V	C
Not Affected			

Example 3-1. LOCAL_CSR_RD

Here is an example of how to access the read data:

```
local_csr_rd[timestamp_low]
immed[gpr_n, 0]
```

3.2.36 LOCAL_CSR_WR

Write specified Microengine CSR register with the data in the specified source register. There is always a 3 cycle delay between local_csr_wr and the value changed.

Instruction Format

local_csr_wr[local_csr, src]

Parameter Descriptions

Parameter	Description	
local_csr	Specifies the Local CSR. See Section 5.2, "Microengine Local CSRs" for a description of the CSRs.	
	CSR Names	
	CTX_ARB_CNTL	BYTE_INDEX
	CTX_ENABLES	T_INDEX
	CC_ENABLE	T_INDEX_BYTE_INDEX
	CSR_CTX_POINTER	INDIRECT_FUTURE_COUNT_SIGNAL
	INDIRECT_CTX_STS	ACTIVE_FUTURE_COUNT_SIGNAL
	ACTIVE_CTX_STS	NN_PUT
	INDIRECT_CTX_SIG_EVENTS	NN_GET
	ACTIVE_CTX_SIG_EVENTS	TIMESTAMP_HIGH, TIMESTAMP_LOW
	INDIRECT_CTX_WAKEUP_EVENTS	NEXT_NEIGHBOR_SIGNAL
	ACTIVE_CTX_WAKEUP_EVENTS	PREV_NEIGHBOR_SIGNAL
	INDIRECT_CTX_FUTURE_COUNT	SAME_ME_SIGNAL
	ACTIVE_CTX_FUTURE_COUNT	CRC_REMAINDER
	INDIRECT_LM_ADDR_0	PROFILE_COUNT
	INDIRECT_LM_ADDR_0_BYTE_INDEX	PSEUDO_RANDOM_NUMBER
	ACTIVE_LM_ADDR_0	
	ACTIVE_LM_ADDR_0_BYTE_INDEX	
	INDIRECT_LM_ADDR_1	
	INDIRECT_LM_ADDR_1_BYTE_INDEX	
ACTIVE_LM_ADDR_1		
ACTIVE_LM_ADDR_1_BYTE_INDEX		
src	Unrestricted source operand for the operation.	

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.37 MSF (Media Switch Fabric)

Move data between the MSF and the ME. The MSF contains the MSF CSRs, RBUF, and TBUF.

Five command types are provided. The *read*, *write* commands perform the same function as the *read64*, and *write64* and differ only in how the *ref_cnt* is interpreted at the MSF unit. The *read64*, and *write64* commands are provided to allow all 16 context relative transfer registers to be filled or emptied using a single read or write instruction. This is useful when moving data between the RBUF/TBUF and the ME. The *fast_wr* command eliminates the need to pull data from a transfer register during a write operation and therefore reduces the time required to complete the write operations. The data written are bits[31:16] = 0 and bits[15:0] = User Data specified by operands *src_opA* + *src_opB*. The intent of the *fast_wr* command is to quickly write the RBUF_Element_Done and Rx_Thread_Freelist_# CSRs.

Instruction Format

```
msf[cmd, xfer, src_op1, src_op2, ref_cnt], opt_tok
```

Parameter Descriptions

Parameter	Cmd	Description
cmd	read	Read from MSF to a Transfer register. The <i>ref_cnt</i> field is specified in 4-byte words
	write	Write to MSF from a Transfer register. The <i>ref_cnt</i> field is specified in 4-byte words
	read64	Read from MSF to a Transfer register. The <i>ref_cnt</i> field is specified in 8-byte words
	write64	Write to MSF from a Transfer register. The <i>ref_cnt</i> field is specified in 8-byte words
	fast_wr	Write the 16 bit data derived from <i>src_op1</i> and <i>src_op2</i> rather than the transfer registers.
xfer	read, read64	An S or D Transfer register. Note 1
	write, write64	An S Transfer register. IXP28xx Rev B: An S or D Transfer Write register. Note 1
	fast_wr	Not used and should always be "--".
src_op1, src_op2	read, write, read64, write64	Restricted operands that define a byte address. The address is specified by <i>src_op1</i> + <i>src_op2</i> . When the read or write instruction is used, access occur on 4-byte word boundaries so bits[1:0] are ignored by the MSF. When the read64 or write64 instruction is used, access occur on 8-byte word boundaries so bits[2:0] are ignored by the MSF. Refer to Section 5.7 (IXP2800) or Section 5.8 (IXP2400) for a list of MSF registers and their addresses. The RBUF is accessed using a read or read64 command and base address of 0x2000. Refer to Tables 3-36 through 3-38. The TBUF is accessed using a write or write64 command and base address of 0x2000. Refer to Tables 3-36 through 3-38.
	fast_wr	Restricted operands that define the byte address of the CSR and the <i>fast_wr</i> data. The address and data is specified by <i>src_op1</i> + <i>src_op2</i> . The CSR address is in bits 15:0, and the data in bits 31:16. Refer to Section 5.7 (IXP2800) or Section 5.8 (IXP2400) for a list of registers and their addresses.
ref_cnt	read and write,	Reference count in increments of 4-byte words. Valid values are 1 to 8
	read64 and write64,	Reference count in increments of 8-byte words. Valid values are 1 to 8
	fast_wr	Not used and should be omitted from the parameter list.

Parameter Descriptions

opt_tok	read, write, read64, write64	ctx_swap[sig_name] refer to Section 3.1.2.4 .	sig_done[sig_name] refer to Section 3.1.2.4 .
		indirect_ref refer to Section 3.1.2.3.1 .	defer[n] (n = 1 to 2) refer to Section 3.1.2.3.1 .
		ordered	Ensure that the MSF instructions with the order token are completed in order that they are executed. This token is used by the assembler and not by hardware.
		ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3	
	fast_wr	none	
1. In 4 context mode, only the S Transfer registers can be used			

Table 3-31. MSF Indirect Format

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
OV	ME				OV	Ref_Cnt				RES				Xfer Register Address				OV	OV	CTX												

Table 3-32. MSF Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
OV [25]	Override bit for Ref_Cnt field
Ref_Cnt	Overrides the ref_cnt field specified by the instruction. This field supports ref_cnts from 0 to 15 where a value of 0 indicates a transfer of 1 and 15 a transfer of 16. Note that when specifying the Ref_cnt and Xfer Register Address, data can be written into another contexts registers set. If this field is changed, the ref_cnt field in the instruction should be a keyword max_nn (where nn = 1-16). Refer to Section 3.1.2.3.2 . For IXP2400, the maximum number of transfer registers that can be accessed using the read64 and write64 commands is 16. As a result, the valid range of values for ref_cnt for read64 and write64 is 0 to 7 instead of 0 to 15.
RES	Reserved
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127). Whether the register set is the S or D is specified in the instruction and cannot be changed using an indirect.
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

The 8K RBUF and TBUF can be configured to support different size elements which effects the number of elements in the RBUF and TBUF. The following tables show how the addresses can be interpreted as element numbers an offsets into the elements. Two tables are provided for each configuration since the offsets size depends on the instruction and command. Bits [12:0] can be viewed as a byte address and the ignored field represent the byte offset which is ignored by the MSF.

Table 3-33. RBUF / TBUF Offset Address 128 64-Byte Elements

1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
0	0	1	Element number								4-byte aligned offset into element				ignored	
read/write Instruction																

1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
0	0	1	Element number								8-byte aligned offset into element	ignored				
read64/write64 command or dram[rbuf_rd]/dram[tbuf_wr] instruction																

Table 3-34. RBUF / TBUF Offset Address 64 128-Byte Elements

1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
0	0	1	Element number								4-byte aligned offset into element				ignored	
read/write command																

1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
0	0	1	Element number								8-byte aligned offset into element	ignored				
read64/write64 command or dram[rbuf_rd]/dram[tbuf_wr] instruction																

Table 3-35. RBUF / TBUF Offset Address 32 256-Byte Elements

1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
0	0	1	Element number								4-byte aligned offset into element				ignored	
read/write command																

1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
0	0	1	Element number								8-byte aligned offset into element	ignored				
read64/write64 command or dram[rbuf_rd]/dram[tbuf_wr] instruction																

3.2.38 MUL_STEP

Used in a multi-instruction operation to multiply two unsigned numbers. Multiplication is done 8-bits per step. [Example 3-1](#) through [Example 3-4](#) show the sequence to use to multiply different size operands. Operands of less than 32 bits must have 0's into unused leading bits. Note that the final add sets the Condition Codes.

Instruction Format

```
mul_step[A_op, B_op], tok
mul_step[dest, --], tok
```

Parameter Descriptions

Parameter	Description	
A_op	Unrestricted source operand (multiplicand). Used when one of the "step" or "start" tokens are specified.	
B_op	Unrestricted source operand (multiplier).Used when one of the "step" or "start" tokens are specified.	
dest	Unrestricted destination operand. Used when one of the "last" tokens are specified. Note that *n\$index++ can not be used as destination.	
--	Must always be "--" when the "Last" tokens are specified	
tok	24X8_start 16X16_start 32x32_start	Indicates the beginning of a new multiply sequence. Load the a_source and b_source into the multiplier array. A separate token is provided for each type of multiply; 24x8, 16x16, or 32x32.
	24X8_step1 16x16_step1 16x16_step2 32x32_step1 32x32_step2 32x32_step3 32x32_step4	Number of the step. 1 step for 24x8, 2 steps for 16x16, 4 steps for 32x32.
	24X8_last 16X16_last 32x32_last	Final add to produce the low 32-bits of the product.
	32x32_last2	Final add to produce the upper 32-bits of a 64-bit product.

Condition Codes Affected

Optional token	N	Z	V	C
Last	Result[31] == 1	Result[31:0] == 0	Set if signed overflow occurs	Carry out from adder[31]
Last2				
Others	Not Affected			

Example 3-1. 8 x 24 multiply (8 bit number is multiplier)

```
mul_step[multiplicand,multiplier], 24x8_start
mul_step[multiplicand,multiplier], 24x8_step1
mul_step[dest,--], 24x8_last
```

Example 3-2. 16 x 16 multiply

```
mul_step[multiplicand,multiplier], 16x16_start
mul_step[multiplicand,multiplier], 16x16_step1
mul_step[multiplicand,multiplier], 16x16_step2
mul_step[dest,--], 16x16_last
```

Example 3-3. 32 x 32 multiply with 32 bit result

```
mul_step[multiplicand,multiplier], 32x32_start
mul_step[multiplicand,multiplier], 32x32_step1
mul_step[multiplicand,multiplier], 32x32_step2
mul_step[multiplicand,multiplier], 32x32_step3
mul_step[multiplicand,multiplier], 32x32_step4
mul_step[dest,--], 32x32_last
```

NOTE: Note that overflow above 32 bit result will not be detected; if the programmer is not ensured of that based on input operands, use the 64 bit version in next Example.

Example 3-4. 32 x 32 multiply with 64 bit result

```
mul_step[multiplicand,multiplier], 32x32_start
mul_step[multiplicand,multiplier], 32x32_step1
mul_step[multiplicand,multiplier], 32x32_step2
mul_step[multiplicand,multiplier], 32x32_step3
mul_step[multiplicand,multiplier], 32x32_step4
mul_step[dest_low,--], 32x32_last
mul_step[dest_high,--], 32x32_last2
```

3.2.39 NOP

Consume one microcycle without performing any operation and without setting any microengine state.

Instruction Format

nop

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.40 PCI

Move data between the MEs and the PCI CSRs or PCI Bus.

Instruction Format

```
pci [cmd, xfer, src_op1, src_op2, ref_cnt], opt_tok
```

Parameter Descriptions

Parameter	Description	
cmd	read	Read from PCI CSR or PCI Bus to the Transfer registers
	write	Write from the Transfer registers to the PCI CSR or PCI Bus.
xfer	read	An S or D Transfer Read register. Note 1
	write	An S Transfer Write register. IXP28xx Rev B: An S or D Transfer Write register. Note 1
src_op1, src_op2	All cmds	Restricted source operands that define the PCI byte address. The address is specified by src_op1 + src_op2. There are six address spaces that are mapped as shown in Table 3-36 .
ref_cnt	All cmds	Reference count in increments of 4 byte words. Valid values are 1 to 8 for the PCI memory space and 1 for all other address spaces.
opt_tok	All cmds	ctx_swap[sig_name] refer to Section 3.1.2.4 . sig_done[sig_name] refer to Section 3.1.2.4 .
		indirect_ref refer to Section 3.1.2.3.1 . defer[n] (n = 1 to 2) refer to Section 3.1.2.3.1 .
		ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3
1. In 4 context mode, only the S Transfer registers can be used		

Table 3-36. PCI Address Space

Address Space	Byte Address Range ¹		Max Ref_cnt size
PCI Memory Space ²	0x2000 0000	0x3FFF FFFF	8
PCI Controller Config ³	0x0600 0000	0x06FF FFFF	1
PCI Controller CSRs ³	0x0700 0000	0x07FF FFFF	1
PCI IACK (reads)	0x0400 0000	0x05FF FFFF	1
Special Cycles (writes)			1
PCI Configuration Cycles (Type 1)	0x0300 0000	0x03FF FFFF	1
PCI Configuration Cycles (Type 0)	0x0200 0000	0x02FF FFFF	1
PCI I/O Cycles ²	0x0000 0000	0x01FF FFFF	1
<p>Note 1: All addresses are specified as byte addresses however the PCI Controller ignores bits[1:0] to create an address on a 4-byte word boundary.</p> <p>Note 2: The address places on the PCI bus is derived from the PCI_ADDR_EXT register concatenated with the address specified in the src_op1 + src_op2.</p> <p>Note 3: The CSR are defined in Section 5.9.</p>			

Table 3-37. PCI Indirect Format

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
OV	ME				OV	Ref_Cnt				OV	RES				Byte Mask				Xfer Register Address				OV	OV	CTX					

Table 3-38. PCI Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
OV [25]	Override bit for Ref_Cnt field
Ref_Cnt	Overrides the ref_cnt field specified by the instruction. This field supports ref_cnts from 0 to 15 where a value of 0 indicates a transfer of 1 and 15 a transfer of 16. Note that when specifying the Ref_cnt and Xfer Register Address, data can be written into another contexts registers set. If this field is changed, the ref_cnt field in the instruction should be a keyword max_nn (where nn = 1-16). Refer to Section 3.1.2.3.2 .
OV [20]	Override bit for Byte Mask field
Byte Mask	Selects which bytes will be written to PCI Bus during a write operation. A “1” signifies the byte will be written. This field is only valid when Ref_Cnt is set to transfer of 1.
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127). Whether the register set is the S or D is specified in the instruction and cannot be changed using an indirect.
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.41 POP_COUNT

Find number of '1' bits in the src register. To find the number of '1's in a partial field, precede the pop_count instruction with an ALU AND instruction to mask off undesired bits. If there are no bits set, the Z condition code is set; otherwise the Z condition code is cleared. This function is implemented in three consecutive instructions, pop_count1, pop_count2, and pop_count3 that must be executed consecutively, with each specifying the same source operand and only the third instruction specifying the destination operand. This instruction is available for The IXP28xx Rev B only.

Instruction Format

```
pop_count1[src]
pop_count2[src]
pop_count3[dest, src]
```

Parameter Descriptions

Parameter	Description
dest	Unrestricted destination that receives the result of the operation. dest[5:0] receives a value from 0 to 32 indicating the number of '1' bits in the src. If there are no bits set, the Z condition code is set; otherwise the Z condition code is cleared. All other bits of dest receive '0'.
src	Unrestricted operand for the operation (B operand). The source has to be the same for all three instructions

Condition Codes Affected

N	Z	V	C
Cleared	Set if src == 0	Cleared	Cleared

3.2.42 RTN

Unconditional branch to the address contained in the lower 12 bits of the specified register. Typically used to return from a branch or jump instruction. For subroutine definitions refer to [Section 2.12](#) and for register lifetime details refer to [Section 2.8.5](#).

Instruction Format

<code>rtn[reg], opt_tok</code>

Parameter Descriptions

Parameter	Description
<code>reg</code>	Unrestricted source that contains the return address. The return address is typically loaded into the register using the <code>load_addr</code> instruction.
<code>opt_tok</code>	<code>defer[n]</code> (n= 1 to 3) refer to Section 3.1.4 .

Condition Codes Affected

N	Z	V	C
Not Affected			

Example 3-1. jump and rtn

```
load_addr[rtn_reg, rtn_label#]
br[sub_routine#]
rtn_label#:

.subroutine
sub_routine#:
; ----- do some work here
rtn[rtn_reg]
.endsub
```

3.2.43 SCRATCH (Read & Write)

Move data between the MEs and scratch memory.

Instruction Format

```
scratch[cmd, xfer, src_op1, src_op2, ref_cnt], opt_tok
```

Parameter Descriptions

Parameter	Description	
cmd	read	Read scratch memory starting at the specified address into the specified transfer registers.
	write	Write scratch memory starting at the specified address from the specified transfer registers.
xfer	read	S or D Transfer Read register. Note 1
	write	S-Transfer Write register. IXP28xx Rev B: An S or D Transfer Write register. Note 1
src_op1, src_op2	All cmds	Restricted operands that define the byte address. The address is specified by src_op1 + src_op2. The scratch memory controller accepts a 32-bit address, however only the low 14-bits are used to address the 16Kbytes of physical memory (address 0x0 to 0x3FFF). Note that Scratch memory is always accessed on 4-byte word boundaries and the scratch memory controller ignores bits[1:0] of the address.
ref_cnt	All cmds	Reference count in increments of 4 byte words. Valid values are 1 to 8.
opt_tok	read, write	ctx_swap[sig_name] refer to Section 3.1.2.4 . sig_done[sig_name] refer to Section 3.1.2.4 .
		indirect_ref refer to Section 3.1.2.3.1 . defer[n] (n = 1 to 2) refer to Section 3.1.2.3.1 .
		ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3
1. In 4 context mode, only the S Transfer registers can be used		

Table 3-39. Scratch (Read and Write) Indirect Format

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
OV	ME					OV	Ref Cnt					RES					Xfer Register Address					OV	OV	CTX					

Table 3-40. Scratch (Read and Write) Indirect Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
OV [25]	Override bit for Ref_Cnt field

Table 3-40. Scratch (Read and Write) Indirect Field Definitions

Field	Description
Ref_Cnt	Overrides the ref_cnt field specified by the instruction. This field supports ref_cnts from 0 to 15 where a value of 0 indicates a transfer of 1 and 15 a transfer of 16. Note that when specifying the Ref_cnt and Xfer Register Address, data can be written into another contexts registers set. If this field is changed, the ref_cnt field in the instruction should be a keyword max_nn (where nn = 1-16). Refer to Section 3.1.2.3.2 .
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127). Whether the register set is the S or D is specified in the instruction and cannot be changed using an indirect.
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.44 SCRATCH (Atomic Operations)

Issue a reference to perform an atomic operation on data in Scratch memory. An atomic operation is one in which a read, modify, and write operation is performed on the memory location and it is assured that another operation will not be allowed to access the data while the atomic operation is in progress.

Instruction Format

```
scratch[cmd, xfer, src_op1, src_op2],opt_tok
```

Parameter Descriptions

Parameter	Cmd	Description
cmd	swap	Swap the contents of a transfer register with the data at the address.
	set	Set the bit(s) at the specified address according to a bit mask provided in the transfer register.
	clr	Clear the bit(s) at the specified address according to a bit mask provided in the transfer register.
	incr	Increment the value of the data at a the specified address by 1. The value in memory rolls over to 0 after 0xFFFF FFFF. The transfer register is not used and should be --.
	decr	Decrement the value of the data at the specified address by 1. The value in memory saturates at 0x0000 0000. The transfer register is not used and should be --.
	add	Add the value in the transfer register to the data at the specified address. Results > 0xFFFF FFFF will roll over.
	sub	Subtract the value in the transfer register to the data at the specified address. Results < 0 will saturate at 0x0000 0000.
	test_and_set	Same as the set instruction, but also return the premodified value to the transfer register that contained the bit mask.
	test_and_clr	Same as the clr instruction, but also return the premodified value to the transfer register that contained the bit mask.
	test_and_incr	Same as the incr instruction, but also return the premodified value to the transfer register that contained the incr value.
	test_and_decr	Same as the decr instruction, but also return the premodified value to the transfer register that contained the decr value.
	test_and_add	Same as the add instruction, but also return the premodified value to the transfer register that contained the add value.
	test_and_sub	Same as the sub instruction, but also return the premodified value to the transfer register that contained the add value.
xfer	incr, decr	Not used and should be "--".
	incr_and_test, decr_and_test	An S or D Transfer Read register that holds the premodified data. Note 1
	Swap & all other test cmds	S-Transfer Write register that holds the data modifier and the S-Transfer Read register that holds the premodified data. IXP28xx Rev B: An S or D Transfer Write and Read register. Note 1
	set, clr, add,sub,	An S-Transfer Write register that holds the data modifier. IXP28xx Rev B: An S or D Transfer Write register. Note 1
src_op1, src_op2	All cmds	Restricted operands that define the byte address. The address is specified by src_op1 + src_op2. The scratch memory controller accepts a 32-bit address, however only the low 14-bits are used to address the 16Kbytes of physical memory (address 0x0 to 0x3FFF). Note that Scratch memory is always accessed on 4-byte word boundaries and the scratch memory controller ignores bits[1:0] of the address.

3.2.45 SCRATCH (Ring Operations)

Issue SCRATCH Ring put or get command to SCRATCH Memory. Sixteen Scratch Rings are supported and are configured via CAP CSRs. The put command puts data onto the Ring while the Get command gets data from the Ring. The number of 4-byte words moved between the ME and the Ring is specified by the `ref_cnt` field. If there is not enough data on the ring to satisfy the `ref_cnt`, only one 4-byte word will be returned, regardless of the value of the `ref_cnt` field, and its value will be all zeroes.

The first 12 rings (0 -11) support a full input state that is set when the ring is full. The MEs should test the state using the `br_inp_state` or `br_linp_state` instructions prior to putting anything on the Ring. The other four rings do not support a full input state and must use another method to indicate there is room available in the Ring, for example an ack signal from the thread that removes data, or a credit counter scheme.

Instruction Format

```
scratch[cmd, xfer, src_op1, src_op2, ref_cnt], opt_tok
```

Parameter Descriptions

Parameter	Cmd	Description
cmd	get	Get the data from the ring specified in the address and return it to the specified transfer registers
	put	Put the data onto the ring specified in the address from the specified SRAM transfer registers
xfer	get	S or D Transfer Read register. Note 1
	put	S Transfer Write register. IXP28xx Rev B: An S or D Transfer Write register. Note 1
src_op1, src_op2	All Cmds	Restricted operands that are added (<code>src_op1 + src_op2</code>) to define the Scratch ring address (0-15). Bits[1:0] are ignored. Refer to Table 3-43 for valid Ring number encodings.
ref_cnt	All Cmds	Reference count. Specifies the number of transfers (1 to 8) in increments of 4 byte words. valid values are 1-8.
opt_tok	All Cmds	ctx_swap[sig_name] refer to Section 3.1.2.4 . sig_done[sig_name] refer to Section 3.1.2.4 .
		indirect_ref refer to Section 3.1.2.3.1 . defer[n] (n = 1 to 2) refer to Section 3.1.4 .
		ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3
	get	ignore_data_error
1. In 4 context mode, only the S Transfer registers can be used		

Table 3-43. SCRATCH Ring Number Encoding (`src_op1 + sr_op2`)

Ring Number	addr value						
0	0	4	0x10	8	0x20	12	0x30
1	4	5	0x14	9	0x24	13	0x34
2	8	6	0x18	10	0x28	14	0x38
3	0xc	7	0x1c	11	0x2c	15	0x3c

Table 3-44. SCRATCH Ring Indirect Format

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
OV	ME				OV	Ref Cnt				RES										Xfer Register Address				OV	OV	CTX				

Table 3-45. SCRATCH Ring Indirect Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
OV [25]	Override bit for Ref_Cnt field
Ref_Cnt	Overrides the ref_cnt field specified by the instruction. This field supports ref_cnts from 0 to 15 where a value of 0 indicates a transfer of 1 and 15 a transfer of 16. Note that when specifying the Ref_cnt and Xfer Register Address, data can be written into another contexts registers set.
RES	Reserved
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127). Whether the register set is the S or D is specified in the instruction and cannot be changed using an indirect.
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

3.2.46 SRAM (Read & Write)

Move data between the MEs and SRAM memory

Instruction Format

```
sram[cmd, xfer, src_op1, src_op2, ref_cnt],opt_tok
```

Parameter Descriptions

Parameter	Cmd	Description	
cmd	read	Read sram memory starting at the specified address into the specified transfer registers.	
	write	Write sram memory starting at the specified address from the specified transfer registers.	
xfer	read	S or D Transfer Read register. Note 1	
	write	S Transfer Write register. IXP28xx Rev B: An S or D Transfer Write register. Note 1	
src_op1, src_op2	All cmds	Restricted operands that define the byte address. The address is specified by src_op1 + src_op2. Bits[1:0] of the address is ignored by the SRAM channels The base addresses of the SRAM channels are Channel 0: 0x0000 0000 Channel 2: 0x8000 0000 Channel 1: 0x4000 0000 Channel 3: 0xc000 0000 (Channels 2 & 3 are Reserved on IXP2400)	
ref_cnt	All cmds	Reference count n increments of 4 byte words. Valid values are 1 to 8.	
opt_tok	read	ignore_data_error	
	All cmds	ctx_swap[sig_name] refer to Section 3.1.2.4 .	sig_done[sig_name] refer to Section 3.1.2.4 .
		indirect_ref refer to Section 3.1.2.3.1 .	defer[n] (n = 1 to 2) refer to Section 3.1.4 .
		ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3	
1. In 4 context mode, only the S Transfer registers can be used			

Table 3-46. SRAM (Read and Write) Indirect Format

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
OV	ME				OV	Ref Cnt	OV	RES	Byte Mask				Xfer Register Address				OV	OV	CTX										

Table 3-47. SRAM (Read and Write) Indirect Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
OV [25]	Override bit for Ref_Cnt field

Table 3-47. SRAM (Read and Write) Indirect Field Definitions

Field	Description
Ref_Cnt	Overrides the ref_cnt field specified by the instruction. This field supports ref_cnts from 0 to 15 where a value of 0 indicates a transfer of 1 and 15 a transfer of 16. Note that when specifying the Ref_cnt and Xfer Register Address, data can be written into another contexts registers set. If this field is changed, the ref_cnt field in the instruction should be a keyword max_nn (where nn = 1-16). Refer to Section 3.1.2.5
OV [20]	Override bit for Byte Mask field
Byte Mask	Selects which bytes will be written to SRAM memory during a write operation. A "1" signifies the byte will be written where bit 12 corresponds to bits 7:0 and bit 15 corresponds to bits 31:24. The Byte Mask field is valid only when Ref_cnt is set to a transfer of 1.
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127). Whether the register set is the S or D is specified in the instruction and cannot be changed using an indirect.
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.47 SRAM (Atomic Operations)

Issue a memory reference to perform an atomic operation on data in SRAM memory. An atomic operation is one in which a read, modify, and write operation is performed on the memory location and it is assured that another atomic operation will not be allowed to access the data while the atomic operation is in progress. Note that non-atomic operations, such as a sram[read], can occur in the middle of a atomic operation.

The IXP28xx Rev B supports no_pull atomic operations for the swap, set, clr, add, test_and_set, test_and_clr, and test_and_add commands. The no_pull version is specified by using the no_pull and the indirect_ref tokens. The indirect reference also provides the write data, which is normally “pulled” from the write transfer register. Also, the no_pull version of the instructions use one less signal.

By not using a transfer register for the data modifier, the SRAM unit does not have to pull any data from the Microengine, which in turn speeds up the execution of the no_pull atomic operations in the SRAM unit.

Instruction Format

```
sram[cmd, xfer, src_op1, src_op2],opt_tok
```

Parameter Descriptions

Parameter	Cmd	Description
cmd	swap	Swap the contents of a transfer register with the data at the address. Note 2
	set	Set the bit(s) at the specified address according to a bit mask provided in the transfer register. A one in the bit position of the bit mask signifies that the bit should be set.
	clr	Clear the bit(s) at the specified address according to a bit mask provided in the transfer register. A one in the bit position of the bit mask signifies that the bit should be cleared.
	incr	Increment the value of the data at a the specified address by 1. The value in memory rolls over to 0 after 0xFFFF FFFF.
	decr	Decrement the value of the data at the specified address by 1. The value in memory saturates at 0x0000 0000.
	add	Add the 2s complement value in the transfer register to the data at the address, which is treated as an unsigned number. Negative number addition whose results is < 0 will saturate the result in the value in memory at 0x0000 0000. Positive number addition whose results is > 0xFFFF FFFF will roll over.
	test_and_set	Same as the set instruction, but also return the premodified value to the transfer register that contained the bit mask. Note 2
	test_and_clr	Same as the clr instruction, but also return the premodified value to the transfer register that contained the bit mask. Note 2
	test_and_incr	Same as the incr instruction, but also return the premodified value to the transfer register that contained the incr value. The premodified value can be used to test for saturation.
	test_and_decr	Same as the decr instruction, but also return the premodified value to the transfer register that contained the decr value. The premodified value can be used to test for saturation.
	test_and_add	Same as the add instruction, but also return the premodified value to the read transfer register specified. The premodified value can be used to test for saturation. Note 2

Parameter Descriptions

Parameter	Cmd	Description
no_pull cmd Note 1	swap	Replace the data at the address with the sign extended contents of the data field from the no_pull indirect reference (Table 3-50). The original data at the address is placed in the read transfer register.
	set	Set one bit at the specified address. The bit to set is in the data field [4:0] specified in the no_pull indirect reference. (Table 3-50). Data field [10:5] is ignored.
	clr	Clear one bit at the specified address. The bit to clear is in the data field [4:0] specified in the no_pull indirect reference (Table 3-50). Data field [10:5] is ignored.
	add	Add the sign extended value in the data field of the no_pull indirect reference (Table 3-50) to the data at the address. Saturates at 0x00000000 if the data field is negative number and the addition results is <0. If the data field is a positive number, then addition results > 0xFFFFFFFF will roll over.
	test_and_set	Same as the set instruction, but also return the premodified value to a read transfer register.
	test_and_clr	Same as the clr instruction, but also return the premodified value to a read transfer register.
	test_and_add	Same as the add instruction, but also return the premodified value to the read transfer register.
xfer	incr, decr	Not used and should be "--".
	incr_and_test, decr_and_test	An S or D Transfer Read register that holds the premodified data. Note 3
	swap & all other test cmds	S-Transfer Write register that holds the data modifier and the S-Transfer Read register that hold the premodified data. IXP28xx Rev B: An S or D Transfer Write and Read register. Note 3
	set, clr, add	An S-Transfer Write register that holds the data modifier. IXP28xx Rev B: An S or D Transfer Write register. Note 3
	set, clr, add, with no_pull opt_tok (IXP28xx Rev B)	Not used and should be "--".
	swap, test_and_set, test_and_clr, test_and_add with no_pull opt_tok (IXP28xx Rev B)	An S or D Transfer Read register to hold the premodified data. Note 3
src_op1, src_op2	All cmds	Restricted operands that define the byte address. The address is specified by src_op1 + src_op2. Bits[1:0] of the address is ignored by the SRAM channels The base addresses of the SRAM channels are Channel 0: 0x0000 0000 Channel 2: 0x8000 0000 Channel 1: 0x4000 0000 Channel 3: 0xc000 0000 (Channels 2 & 3 are Reserved on IXP2400)

Parameter Descriptions

Parameter	Cmd	Description		
opt_tok	test_and_incr, test_and_decr set, clr, add,	sig_done[sig_name] refer to Section 3.1.2.4.	ctx_swap[sig_name] refer to Section 3.1.2.4.	
		indirect_ref refer to Section 3.1.2.3.1.	defer[n] (n = 1 to 2) refer to Section 3.1.4.	
		ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3		
	test_and_set, test_and_clr, test_and_add, swap,	sig_done[sig_name2] refer to Section 3.1.2.4.	ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3	
		indirect_ref refer to Section 3.1.2.3.1.		
	incr, decr	indirect_ref refer to Section 3.1.2.3.1.	ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3	
	set, clr, add, with no_pull opt_tok (IXP28xx Rev B)	no_pull (required)	indirect_ref (required) refer to Section 3.1.2.3.1.	
		ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3		
	swap, test_and_set, test_and_clr, test_and_add, with no_pull opt_tok (IXP28xx Rev B)	no_pull (required)	indirect_ref (required) refer to Section 3.1.2.3.1.	
		ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3	sig_done[sig_name] refer to Section 3.1.2.4.	
ctx_swap[sig_name] refer to Section 3.1.2.4.				
<p>1. The B0 revision of the IXP28x0 supports a “no_pull” version of these commands. The no_pull version is specified by the indirect reference token no_pull. The indirect reference also provides the write data, which is normally “pulled” from the write transfer register. The no_pull commands do not require a write transfer register.</p> <p>2. Two signals are required and this affects how the BR_SIGNAL, BR_!SIGNAL instructions are used. Refer to the BR_SIGNAL, BR_!SIGNAL instructions for details.</p> <p>3. In 4 context mode, only the S Transfer registers can be used</p>				

Table 3-48. SRAM Indirect Format (IXP28xx Rev A: all Atomics; IXP28xx Rev B: Pull Atomics)

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
OV	ME				RES				OV	RES	Byte Mask				Xfer Register Address				OV	OV	CTX										

Table 3-49. SRAM Indirect Field Definitions (IXP28xx Rev A: all Atomics; IXP28xx Rev B: Pull Atomics)

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3.
RES	Reserved and must be zero
OV [20]	Override bit for Byte Mask field

Table 3-49. SRAM Indirect Field Definitions (IXP28xx Rev A: all Atomics; IXP28xx Rev B: Pull Atomics)

Field	Description
Byte Mask	Selects which bytes will be written to SRAM memory during a write operation. A "1" signifies the byte will be written.
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127).
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Table 3-50. SRAM Indirect Format (IXP28xx Rev B: no_pull Atomics)

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
OV	ME				OV	upper no-pull data	OV	lower no-pull data				Xfer Register Address				OV	OV	CTX													

Table 3-51. SRAM Indirect Field Definitions (IXP28xx Rev B: no_pull Atomics)

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
OV [25:24]	Override bits for no-pull mode. Both bits and bit [20] must be set to enable the no-pull mode.
upper no_pull data	Bits[10:8] of the no_pull data field.
OV [20]	Override bit for no-pull mode. This bit and bits [25:24] must be set to enable the no-pull mode.
Lower no_pull data	Bits[7:0] of the no_pull data field.
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127).
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.48 SRAM (CSR)

Issue a reference to read or write the SRAM channel Control and Status Registers.

Instruction Format

```
sram[cmd, xfer, src_op1, src_op2],opt_tok
```

Parameter Descriptions

Parameter	Cmd	Description
Command	csr_rd	Read the SRAM CSR specified by the address and put the data into the specified transfer register.
	csr_wr	Write the SRAM CSR specified by the address with the data in the specified transfer register.
xfer	csr_rd	S or D Transfer Read register. Note 1
	csr_wr	S Transfer Write register. IXP28xx Rev B: An S or D Transfer Write register. Note 1
src_op1, src_op2	All cmds	Restricted operands that define the byte address. The address is specified by src_op1 + src_op2. Bits[1:0] of the address is ignored by the SRAM channels The offsets to the CSRs are shown in Table 5-16 . The base address for the SRAM channels are: Channel 0: 0x0000 0000 Channel 1: 0x4000 0000 Channel 2: 0x8000 0000 Channel 3: 0xC000 0000 (Channels 2 & 3 are Reserved on IXP2400)
opt_tok	All cmds	ctx_swap[sig_name] refer to Section 3.1.2.4 .
		sig_done[sig_name] refer to Section 3.1.2.4
		indirect_ref refer to Section 3.1.2.3.1 .
		defer[n] (n = 1 to 2) refer to Section 3.1.4 .
		ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3
1. In 4 context mode, only the S Transfer registers can be used		

Table 3-52. SRAM CSR Indirect Format

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
↓\O	ME				RES												Xfer Register Address				↑\O	↓\O	CTX							

Table 3-53. SRAM CSR Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
RES	Reserved
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127). Whether the register set is the S or D is specified in the instruction and cannot be changed using an indirect.
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.49 SRAM (Read Queue Descriptor)

Issue a memory reference to an SRAM Channel to read the Queue descriptor into the SRAM Queue Array (see [Figure 3-2](#). Three commands are provided to read the head and q_count, the tail and q_count, or the other (which reads the head if it is not currently in the array or the tail if it is currently not in the array). Optionally, additional data (as defined by the application) that begins at the specified address + 3 can be read from the SRAM into the ME transfer registers.

Instruction Format

```
sram[cmd, xfer, src_op1, src_op2, ref_cnt], opt_tok
```

Parameter Descriptions

Parameter	Cmd	Description
cmd	rd_qdesc_head	This command is used to partially load a new queue_descriptor into the queue_array at an assigned entry number. Dequeues can be performed once the head is loaded. Follow rd_qdesc_head with rd_qdesc_other to completely load the queue_array entry. Refer to Figure 3-2 for a description of fields moved between the queue descriptor, SRAM Queue Array and ME
	rd_qdesc_tail	This command is used to partially load a new queue_descriptor into the queue_array at an assigned entry number. Enqueues can be performed once the tail is loaded. Follow rd_qdesc_tail with rd_qdesc_other to completely load the queue_array entry. Refer to Figure 3-2 for a description of fields moved between the queue descriptor, SRAM Queue Array and ME
	rd_qdesc_other	This command is used to finish loading a new queue_descriptor into the queue_array at an assigned entry number. This command should only be used when either the head or tail is valid in the same queue_array entry. Refer to Figure 3-2 for a description of fields moved between the queue descriptor, SRAM Queue Array and ME
xfer	rd_qdesc_head rd_qdesc_tail	S or D Transfer Read register where the q_count and optional data is returned. Note 1
	rd_qdesc_other	Must be --
src_op1, src_op2	All Cmd	Restricted operands that are added (src_op1 + src_op2) to define the following: [31:30] SRAM Channel [29:24] Queue Array Entry Number [23:0] Queue Descriptor data block The Address of Queue Descriptor block specifies a 4-byte word address (not a byte address) that must be the start of the 16-byte aligned address of the Queue descriptor (i.e bits [1:0] should always be 0). The queue descriptor consists of the head, tail, q_count and optional data.
ref_cnt	rd_qdesc_head rd_qdesc_tail	Reference count n increments of 4 byte words. Valid values are 2 to 8. The number of words returned to transfer registers is one less than the ref_cnt specified. For example, if the ref_cnt were "3", then three words would be read from sram, the first would be used by the SRAM Queue Array hardware, and the last two (q_count and optional word) would be returned to transfer registers.
	rd_qdesc_other	Not required and must be omitted from the parameter list

Parameter Descriptions

Parameter	Cmd	Description	
opt_tok	All Cmds	indirect_ref refer to Section 3.1.2.3.1 .	ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3
	rd_qdesc_head	sig_done[sig_name] refer to Section 3.1.2.4 .	ctx_swap[sig_name] refer to Section 3.1.2.4 .
	rd_qdesc_tail	defer[n] (n = 1 to 2) refer to Section 3.1.2.3.1 .	ignore_data_error
1. In 4 context mode, only the S Transfer registers can be used			

Figure 3-2. Read Queue Descriptor Commands

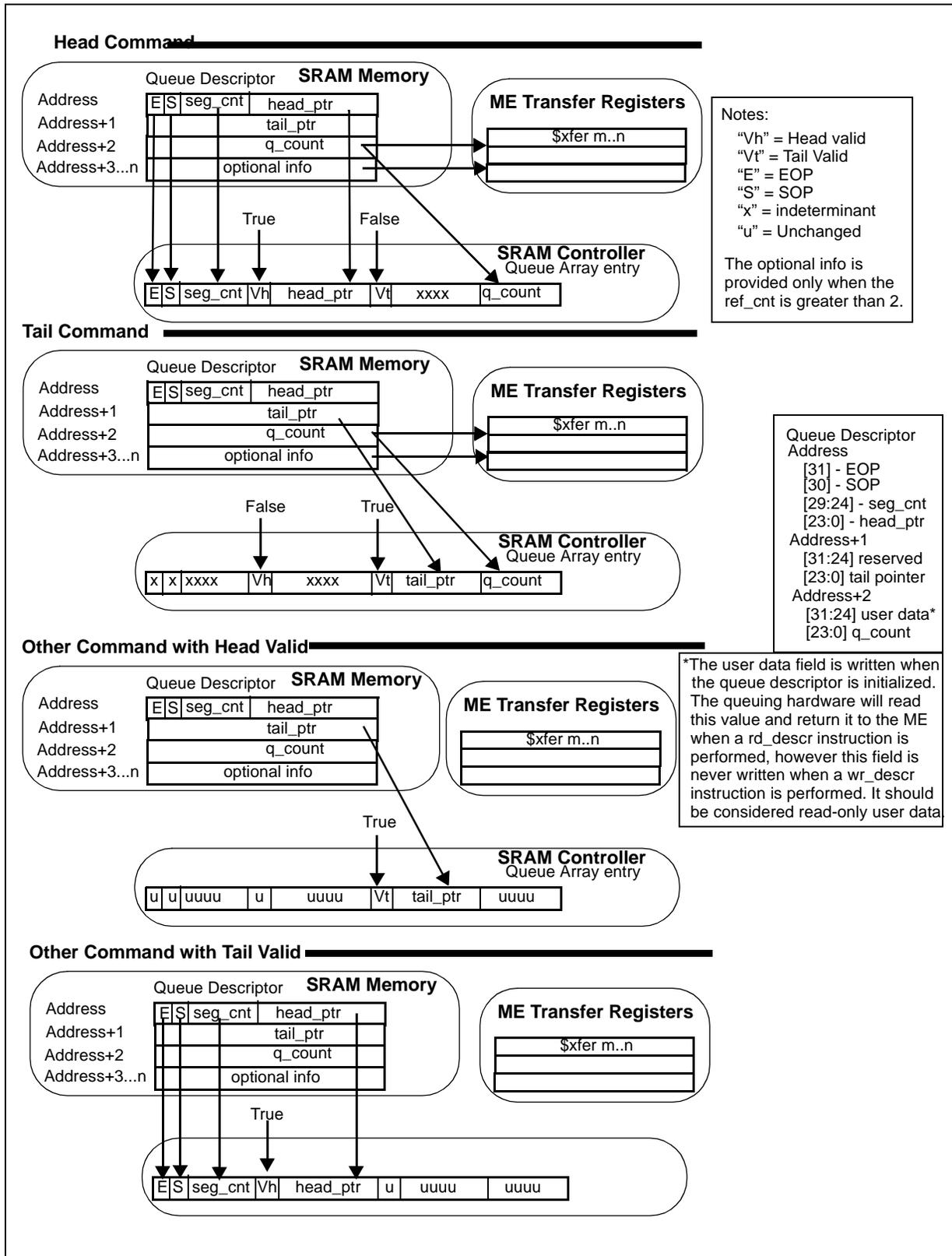


Table 3-54. SRAM (Read Queue Descriptor) Indirect Format

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
OV	ME				OV	Ref Cnt			RES						Xfer Register Address				OV	OV	CTX										

Table 3-55. SRAM (Read Queue Descriptor) Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
OV [25]	Override bit for Ref_Cnt field
Ref_Cnt	Overrides the ref_cnt field specified by the instruction. This field supports ref_cnts from 1 to 15 (0 not legal) where a value of 1 indicates a transfer of 1 (q_count) to the ME and 15 a transfer of 15 (q_count and 14 x 4 bytes of optional data). Note that when specifying the Ref_cnt and Xfer Register Address, data can be written into another contexts registers set.
RES	Reserved and must be ZERO
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127). Whether the register set is the S or D is specified in the instruction and cannot be changed using an indirect.
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

3.2.50 SRAM (Write Queue Descriptor)

Issue a memory reference to an SRAM Channel to move data from the SRAM Queue Array into SRAM.

Instruction Format

```
sram[cmd, --, src_op1, src_op2]
```

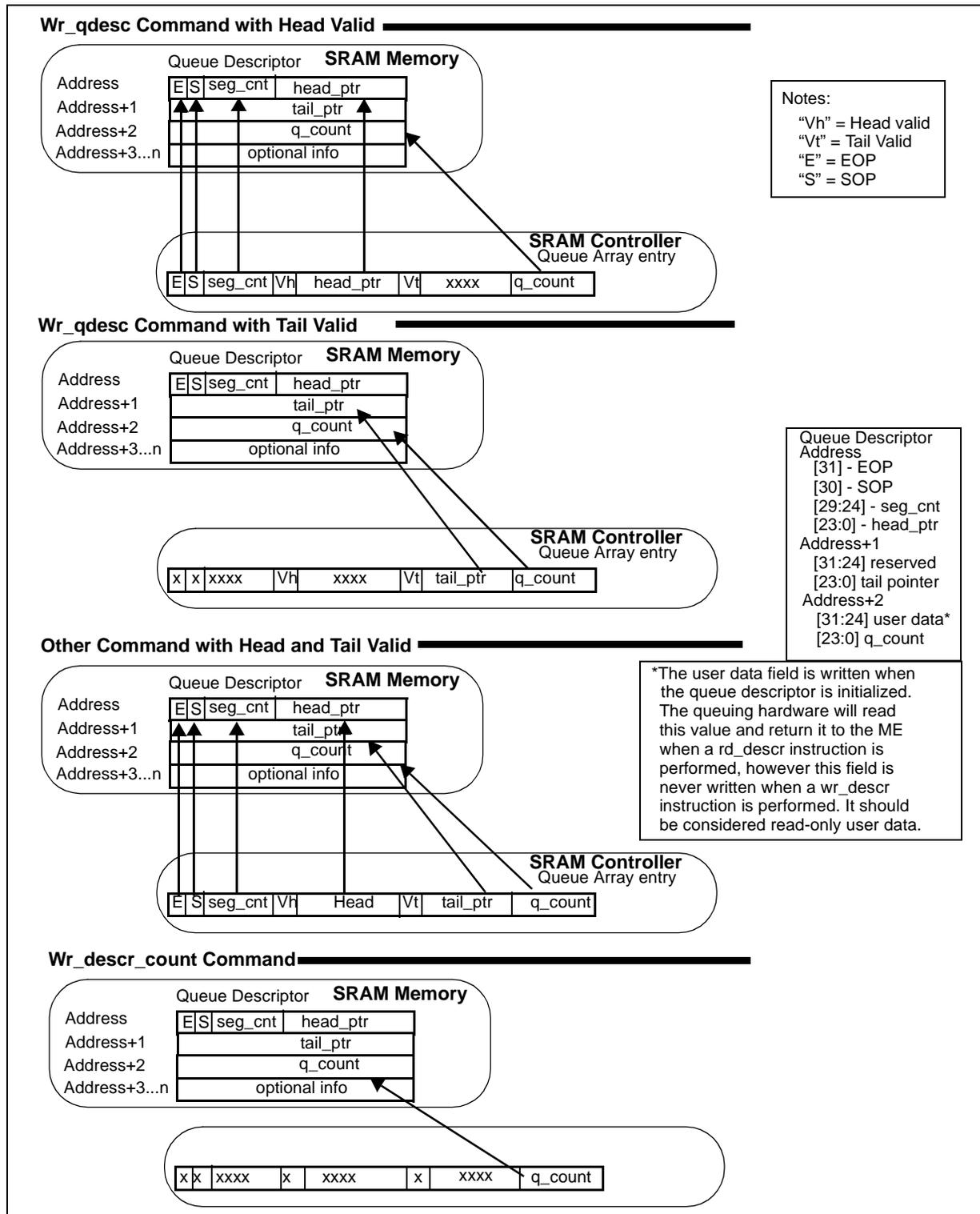
Parameter Descriptions

Parameter	Description	
cmd	wr_qdesc	<p>This command is used to evict an entry in the SRAM Q_array and returns its contents to SRAM memory. Only the fields that are valid in the Q_descriptor are written in order minimize SRAM bandwidth utilization.</p> <p>If the head valid bit is set in the Q_array, the head, seg_cnt, SOP, EOP, and q_count fields are returned to SRAM.</p> <p>If the tail valid bit is set in the Q_array, the tail, and q_count fields are returned to SRAM.</p> <p>If head and tail are valid, the head, seg_cnt, SOP, EOP, tail, and q_count fields are returned to SRAM.</p> <p>If neither the head and tail are valid, nothing is returned to SRAM.</p> <p>Refer to Figure 3-3 for a description of fields moved between the Q_array and the SRAM</p>
	wr_qdesc_count	<p>This command is used to evict the q_count entry in the SRAM Q_array and return its contents to SRAM memory.</p> <p>Refer to Figure 3-3 for a description of fields moved between the Q_array and the SRAM</p>
--	All Cmds	Must always be "--"
src_op1, src_op2	All Cmds	<p>Restricted operands that are added (src_op1 + src_op2) to define the following:</p> <ul style="list-style-type: none"> [31:30] SRAM Channel [29:24] Queue Array Entry Number [23:0] Queue Descriptor data block <p>The Address of Queue Descriptor block specifies a 4-byte word address (not a byte address) that must be the start of the 16-byte aligned address of the Queue descriptor (i.e bits [1:0] always should be 0). The queue descriptor consists of the head, tail, q_count and optional data.</p>

Condition Codes Affected

N	Z	V	C
Not Affected			

Figure 3-3. Write Queue Descriptor Commands



3.2.51 SRAM (Enqueue)

Two commands are provided for performing enqueue operations. The enqueue command is used to enqueue to a queue structure that supports a single buffer per packet (refer to [Figure 3-4](#)). The enqueue_tail (enqueue tail) command is used with the enqueue command for queue structures that support multiple buffers per packet (refer to [Figure 3-4](#) and [Figure 3-5](#)). In this case the enqueue command links the first buffer to the queue array and the enq_tail command updates the tail pointer in the SRAM Queue Array. The default is to set the segment count (seg_cnt) to 0 and set the SOP and EOP bits. This can be overridden using the indirect reference optional token.

This instruction uses src_op1 and src_op2 parameters to select the SRAM channel and SRAM Queue Array entry as well as to pass the address of the buffer descriptor to the SRAM Queue Array. The indirect reference is used to set the EOP, SOP, and seg_cnt fields.

Note:

1. When performing SRAM Enqueues from the Intel XScale® core, the EOP, SOP, & Segment Count fields are always written with values EOP = 1, SOP = 1, & Segment Count = 0.
2. The Intel XScale® core cannot perform enqueue operations if the queue controller is in Mode 3 since a Segment Count of zero on enqueue is illegal in this mode. Refer to [Section 3.2.52, “SRAM \(Dequeue\)”](#) for a description of Mode 3.

Instruction Format

```
sram[cmd, --, src_op1, src_op2],opt_tok
```

Parameter Descriptions

Parameter	Command	Description
cmd	enqueue	This command is used to add a single buffer to the queue or to add the Start-of-Packet buffer of a multi-buffer frame to the queue. It adds a buffer to the queue contained in the Q-array entry, and sets the tail to point to the buffer. If necessary, a link is established from the old tail buffer to the new buffer. If SRAM_CONTROL[QC_IGN_EOP] is set, the value of seg_cnt is added to q_count. Otherwise the q_count is incremented.
	enqueue_tail	This command updates the tail pointer only. This command must be preceded by an enqueue command to the same entry. This adds the End-of-Packet buffer of a multi-buffer frame to the queue. There must be no intervening commands to a queue array entry between an enqueue and enqueue_tail command.
--	All Cmd	Must always be "--"
src_op1, src_op2	All Cmd	Restricted operands that are added (src_op1 + src_op2) to define the following: [31:30] SRAM Channel [29:24] Queue Array Entry Number [23:0] Q_link Pointer The Q_link Pointer must be a 4-byte word address.
opt_tok	All Cmds	indirect_ref refer to Section 3.1.2.3.1 .

Table 3-56. SRAM (Enqueue) Indirect Format

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0				
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0													
RES												OV	EOP	SOP	SEG_CNT										RES									

Table 3-57. SRAM (Enqueue) Field Definitions

Field	Description						
RES	Reserved and must be zero						
OV [20]	Override bit for the EOP, SOP, and SEG_CNT fields						
EOP	This sets the EOP bit in the Q_link.						
SOP	This sets the SOP bit in the Q_link.						
SEG_CNT	<p>This sets the segment count in the Q_link.</p> <p>When the Queue Array HW set to Mode 0 and 1, the segment count encoding is as follows:</p> <table border="0"> <tr> <td>0x00 = 1 segment</td> <td>0x03 = 4 segment</td> </tr> <tr> <td>0x01 = 2 segment</td> <td>to...</td> </tr> <tr> <td>0x02 = 3 segment</td> <td>0x3E = 63 segment</td> </tr> </table> <p>When the Queue Array HW set to Mode 3, the segment count encoding is as follows:</p> <p>0x1 = add 1 buffer to the queue count</p> <p>to...</p> <p>0x3E = add 62 buffers to the queue count.</p>	0x00 = 1 segment	0x03 = 4 segment	0x01 = 2 segment	to...	0x02 = 3 segment	0x3E = 63 segment
0x00 = 1 segment	0x03 = 4 segment						
0x01 = 2 segment	to...						
0x02 = 3 segment	0x3E = 63 segment						
<p>Notes:</p> <ol style="list-style-type: none"> When an indirect ref is not specified, the parameters delivered to the SRAM controller are SOP =1, EOP =1, and SEG_CNT = 0x3F (which is translated by the SRAM controller as a SEG_CNT of 0). If in Queue Array HW is in Mode 1, the segment count field is not used by hardware and can be used by software as a message field between the enqueueer and dequeueer. If in Queue Array HW is in Mode 3, a segment count of 0 is illegal and is considered a programming error if used. 							

Condition Codes Affected

N	Z	V	C
Not Affected			

Figure 3-4. Enqueue One Buffer at a Time using the Enqueue Command

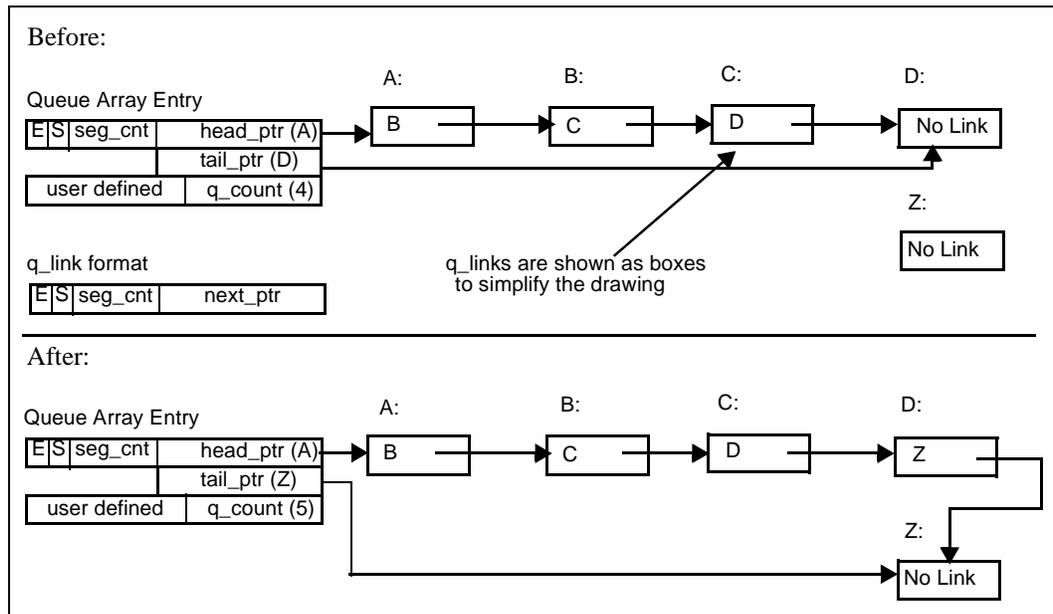
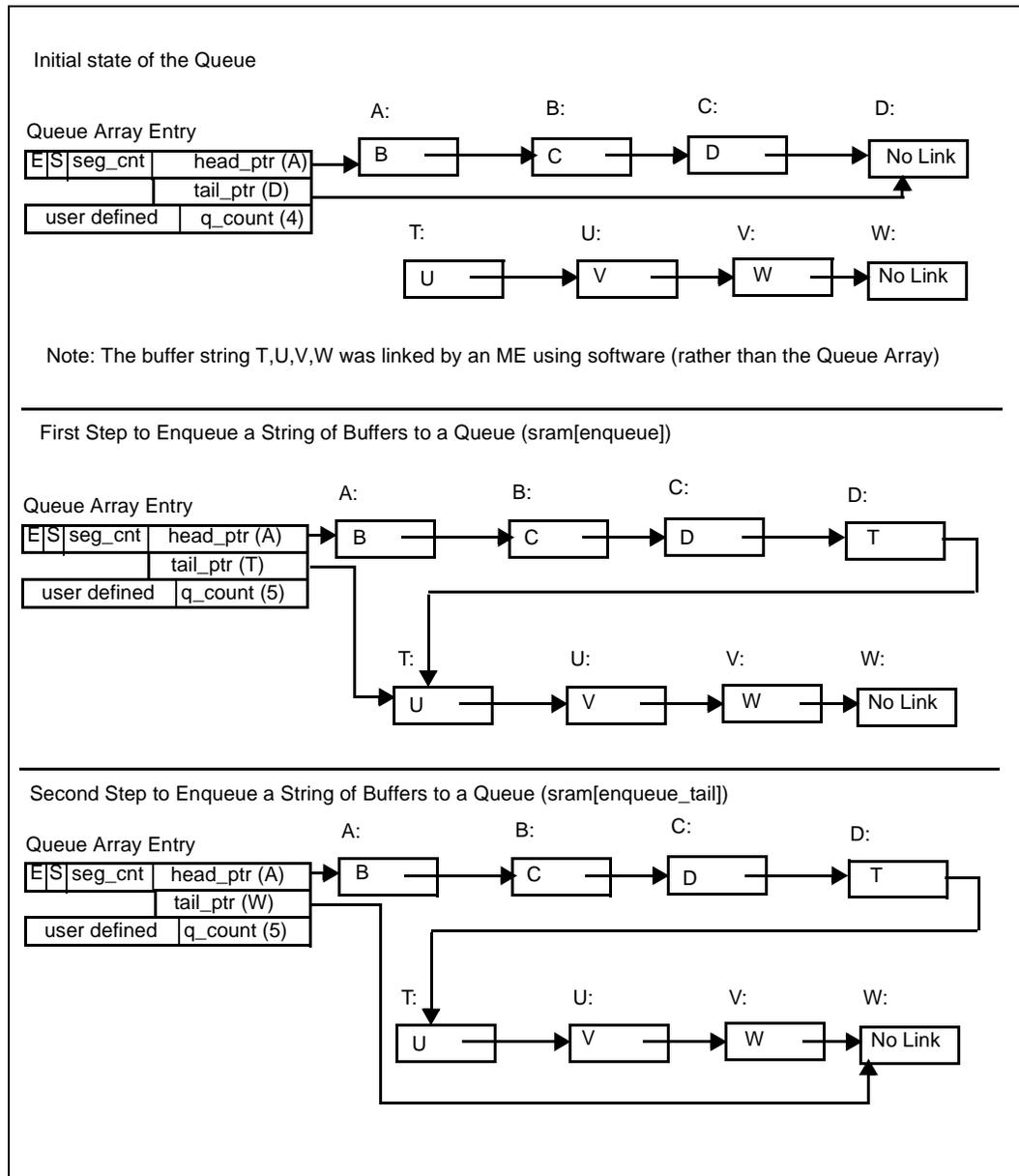


Figure 3-5. Enqueue a String of Buffers to a Queue



3.2.52 SRAM (Dequeue)

The SRAM_CONTROL[QC_IGN_EOP:QC_IGN_SEG_CNT] register bits support the following three modes that determine the behavior of the dequeue command (Mode 2 is not supported). Refer to Figure 3-7 for examples of these modes.

Mode 0: Dequeue Segments & Count Packets

The seg_cnt is decremented for each sram[dequeue] command. Only when seg_cnt equals 0 is the q_link removed from the linked list. The EOP is used by hardware to determine if it should decrement the q_count, therefore it must be set on the last buffer of a packet for software designs that support multiple buffers per packet or on all buffers for software designs that support a single buffer per packet. The state of the EOP bit is returned with the data for each dequeue command. The state of the SOP bit is returned only with the data for the first dequeue command to a buffer. The SOP bit is clear for subsequent dequeue commands to the buffer.

Mode 1: Dequeue Buffers & Count Packets

The seg_cnt is ignored in this mode so a q_link is removed from the linked list for each sram[dequeue] command. The EOP and SOP bits are treated the same as mode 0. The seg_cnt is returned unchanged on each dequeue.

Mode 3: Dequeue Buffers & Count Buffers

The seg_cnt is ignored in this mode so a q_link is removed from the linked list for each sram[dequeue] command. The EOP bit is also ignored by hardware so the q_count is always decremented for each dequeue command. Note: In this mode the seg_cnt is added to the q_count on every enqueue. A seg_cnt value of 0 is illegal for enqueue commands.

Instruction Format

sram[cmd, xfer, src_op1, src_op2], opt_tok

Parameter Descriptions

Parameter	Command	Description
cmd	dequeue	This command is used to remove a q_link from the queue. If the q_count is zero (nothing on the queue), the value of zero is returned. If the q_count is NOT zero (something on the queue), and the seg_cnt is 0 in the head q_link (buffer only has one segment), then the q_link is removed from the linked list and it is returned. If the q_count is NOT zero (something on the queue), and the seg_cnt is NOT 0 in the head q_link (buffer holds multiple segments), then the seg_cnt is decremented and the q_link NOT is removed from the linked list. If the q_count is not 0 then the entire q_link is returned.
xfer	All cmds	S or D Transfer Read register where the q_link is returned. Note 1
src_op1, src_op2	dequeue	Restricted operands that are added (src_op1 + src_op2) to define the following: [31:30] SRAM Channel [29:24] Queue Array Entry Number [23:0] ignored

Parameter Descriptions

Parameter	Command	Description	
opt_tok	dequeue	indirect_ref refer to Section 3.1.2.3.1 .	defer[n] refer to Section 3.1.4 .
		ctx_swap[sig_name] refer to Section 3.1.2.4 .	sig_done[sig_name] refer to Section 3.1.2.4 .
		ind_targets[me1, me2, ...] refer to Section 3.1.2.3.3	
1. In 4 context mode, only the S Transfer registers can be used			

Figure 3-6. Dequeue Buffer

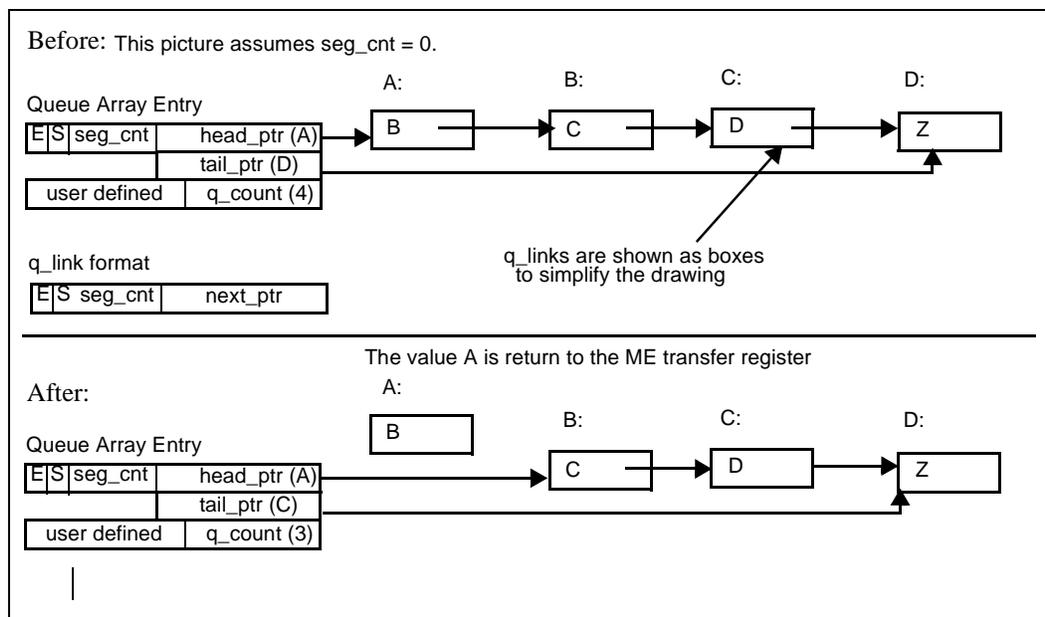


Table 3-58. SRAM (dequeue) Indirect Format

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
OV	ME		RES																	Xfer Register Address				OV	OV	CTX					

Table 3-59. SRAM (dequeue) Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
RES	Reserved
Xfer Register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127). Whether the register set is the S or D is specified in the instruction and cannot be changed using an indirect.

Table 3-59. SRAM (dequeue) Field Definitions

Field	Description
OV [4]	Override bit for Xfer Register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

Condition Codes Affected

N	Z	V	C
Not Affected			

Figure 3-7. Example of the Three Dequeue Modes

Mode 0: Dequeue Segments & Count Packets
 Example: One Packet in 2 buffers occupying 6 segments

seg 1	seg 5	Q_link 1: seg_cnt = 4, decremented each dequeue SOP = 1, cleared on first dequeue EOP = 0, q_count not decremented Q_link is removed when seg_cnt = 0 (4 dequeues)
seg 2	seg 6	
seg 3	empty	
seg 4	empty	
Buffer 1	Buffer 2	Q_link 2: seg_cnt = 2, decremented each dequeue SOP = 0 EOP = 1, q_count decremented when seg_cnt = 0 Q_link is removed when seg_cnt = 0 (2 dequeues)

q_count is incremented each enqueue (not enqueue_tail)

Mode 1: Dequeue Buffers & Count Packets
 Example: One Packet in 2 buffers

data	data	Q_link 1: seg_cnt is ignored SOP = 1 EOP = 0, q_count not decremented Q_link 2: seg_cnt is ignored SOP = 0 EOP = 1, q_count decremented
Buffer 1	Buffer 2	

Q_link is removed on each dequeue
q_count is incremented each enqueue (not enqueue_tail)

Mode 3: Dequeue Buffers & Count Buffers
 Example: One Packet in 2 buffers

data	data	Q_link 1: seg_cnt = 2, this is added to the current q_count during the enqueue (not enqueue_tail) to indicate that two buffers are being enqueued. It is ignored on dequeue. SOP = 1 EOP = ignored, q_count decremented Q_link 2: seg_cnt is ignored SOP = 0 EOP = ignored, q_count decremented
Buffer 1	Buffer 2	

Q_link is removed on each dequeue

3.2.53 SRAM (Ring Operations)

Issue an SRAM Ring put or get command to the SRAM. Each Rings uses one of the 64 SRAM Queue Array Entries in the SRAM controller. Note that the maximum number of SRAM rings is equal to 64 x the number of SRAM channels supported by the Network Processor. Also, note that the ring descriptor must be read from SRAM using the Read Queue Descriptor command before issuing any put or get operations to it. The format of the ring descriptor is shown in table Table 3-60.

Table 3-60. SRAM Ring Descriptor Format

Name	Longword #	Bit #	Definition
Ring/Journal Size	0	31:29	See Table 3-61 for size encoding.
Head	0	23:0	Get pointer
Tail	1	23:0	Put pointer
Ring Count	2	23:0	Number of longwords on the ring
NOTE: For Ring/Journal, Head and Tail must be initialized to the same address			

Journals and Rings can be configured to be one of eight sizes, as shown in Table 3-61.

Table 3-61. SRAM Ring Size Encoding

Ring Size Encoding	Size of Journal/Ring Area	Head/Tail Field Base	Head and Tail Field Increment
000	512 Longwords	23:9	8:0
001	1K	23:10	9:0
010	2K	23:11	10:0
011	4K	23:12	11:0
100	8K	23:13	12:0
101	16K	23:14	13:0
110	32K	23:15	14:0
111	64K	23:16	15:0

The put command puts data onto the Ring while the Get command gets data from the Ring. The number of 4-byte words moved between the ME and the Ring is specified by the `ref_cnt` field. For the Put command, a 4-byte Status word is returned that indicates the current number of 4-byte words on the Ring and whether the put was successful. The put command will be unsuccessful if there are 16 or less free entries on the ring for Rev A and 64 or less free entries for Rev B. The Status word is written to read transfer register specified by `xfer` field. If the Put operation is unsuccessful, the data is dropped (not put onto the ring) and the ME should retry the put command.

The ring area will always starts at an address aligned to the ring size (example if the size is 1K the ring area will start on a 1K address boundary).

The `sram[rd_qdesc]` commands must be used to initialize the queue array entry before the Ring can be used.

Instruction Format

```
sram[cmd, xfer, src_op1, src_op2, ref_cnt], opt_tok
```

Parameter Descriptions

Parameter	Cmd	Description	
cmd	get	Get the data from the ring specified in the address and return it to the specified transfer registers	
	put	Put the data into the ring specified in the address from the specified transfer registers	
xfer	get	S or D Transfer Read register. Note 1	
	put	S-Transfer register. The write transfer registers holds the data that is put onto the ring, the read transfer register holds the Full Status word which has the format: [31] Success = 1, Fail = 0 [30:16] Always 0 [15:0] Number of 4-byte words on the Ring before the PUT operation IXP28xx Rev B: An S or D Transfer Write register. Note 1	
src_op1, src_op2	All Cnds	Restricted operands that are added (src_op1 + src_op2) to define the following: [31:30] SRAM Channel [29:8] ignored [7:2] Ring Number [1:0] ignored	
ref_cnt	All Cnds	Reference count. Specifies the number of transfers (1 to 8) in increments of 4 byte words. If the indirect_ref token is specified, ref_cnt can be a keyword max_nn (where nn = 1-16). Refer to Section 3.1.2.3.2 .	
opt_tok	put	sig_done[sig_name2] refer to Section 3.1.2.4 .	indirect_ref refer to Section 3.1.2.3.1 .
		sig_done[sig_name] refer to Section 3.1.2.4 .	ctx_swap[sig_name] refer to Section 3.1.2.4 .
	get	defer[n] refer to Section 3.1.4	indirect_ref refer to Section 3.1.2.3.1 .
1. In 4 context mode, only the S Transfer registers can be used			

Table 3-62. SRAM Ring Indirect Format

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
OV	ME					OV	Ref Cnt					RES					Xfer Register Address					OV	OV	CTX							

Table 3-63. SRAM Ring Indirect Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	Specifies the Microengine where the result will be written and signaled upon completion [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
OV [25]	Override bit for Ref_Cnt field
Ref_Cnt	Overrides the ref_cnt field specified by the instruction. This field supports ref_cnts from 0 to 15 where a value of 0 indicates a transfer of 1 and 15 a transfer of 16. Note that when specifying the Ref_cnt and Xfer Register Address, data can be written into another contexts registers set.

Table 3-63. SRAM Ring Indirect Field Definitions

Field	Description
Xfer register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127).
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	Specifies the context where the result will be written and signaled upon completion.

3.2.54 SRAM (Journal Operations)

Issue an SRAM Journal command to the SRAM channel. A Journal is similar to a Ring except that only the put operation is supported and the data written to the journal ring will wrap to the beginning when the Journal Ring fills. Data is read from the Ring using a standard sram read instruction. Each Journal Ring uses one of the 64 SRAM Queue Array Entries in the SRAM controller. Note that the maximum number of Journal rings is equal to 64 times the number of SRAM channels supported by the Network Processor. Also, note that the journal descriptor must be read from SRAM using the Read Queue Descriptor command before issuing any journal operations to it. The format of the journal descriptor is similar to that of the ring descriptor shown in table [Table 3-60](#).

The journal command puts data onto the Journal Ring from the transfer registers. The number of 4-byte words put onto the Journal Ring is specified by the `ref_cnt` field. The `fast_journal` command puts immediate data onto the Journal Ring. The `fast_journal` command always writes a tag that tells the SRAM Channel to insert the Cluster, ME and thread numbers into bits [31: 24] of the data. (ME Cluster bit[31], ME number bits [30: 27] and thread number bits [26: 24]).

The journal area will always start at an address aligned to the journal size (example if the size is 1K the journal area will start on a 1K address boundary).

The `sram[rd_qdesc]` commands must be used to initialize the queue array entry before the Journal can be used.

An example use for the journal is for debug:

- The ME threads write to the journal area at specific points in the code. Examples of information written to the journal area might be packet header/timestamp/buffer pointer, tagged with ME name.
- An ME hits a breakpoint based on a error and all the MEs are stopped. For example, when it receives an illegal `buffer_pointer`.
- The breakpoint routine results in all the MEs to be stopped.
- The user reads the journal area to help unravel what went wrong.

Instruction Format

```
sram[cmd, xfer, src_op1, src_op2, ref_cnt],opt_tok
```

Parameter Descriptions

Parameter	Cmd	Description	
cmd	journal	Put the data in the specified transfer registers onto the specified journal ring.	
	fast_journal	Put the data along with a tag onto the specified journal ring. The data has the following format. [31:27] ME number (tag) [26:24] Context number (tag) [23:0] Journal data The ME and context number is that which wrote the data or if the indirect option is used it is the ME and context specified by the fields in the indirect reference. Journal data is specified by src_op1 + src_op2. Note that the journal data is provided with the command and no pull operation is required on the pull bus.	
xfer	journal	S-Transfer Write register the specifies the first of a contiguous set of registers that hold the data that is to be put onto the journal ring. IXP28xx Rev B: An S or D Transfer Write register. Note 1	
	fast_journal	Not required and should be "--"	
src_op1, src_op2	journal	Restricted operands that are added (src_op1 + src_op2) to define the following: [31:30] SRAM Channel [29:24] Queue Array Entry (journal number) [23:0] not used	
	fast_journal	Restricted operands that are added (src_op1 + src_op2) to define the following: [31:30] SRAM Channel [29:24] Queue Array Entry (journal number) [23:0] Journal data	
ref_cnt	journal	Reference count. Specifies the number of transfers (1 to 8) in increments of 4 byte words. If the indirect_ref token is specified, ref_cnt can be a keyword max_nn (where nn = 1-16). Refer to Section 3.1.2.3.2 .	
	fast_journal	Not required and should be omitted from the parameter list	
opt_tok	journal	ctx_swap[sig_name] refer to Section 3.1.2.4 .	defer[n] (n = 1 to 2) refer to Section 3.1.4
		sig_done[sig_name] refer to Section 3.1.2.4 .	indirect_ref refer to Section 3.1.2.3.1 .
	fast_journal	indirect_ref refer to Section 3.1.2.3.1 .	
1. In 4 context mode, only the S Transfer registers can be used			

Table 3-64. SRAM Journal Indirect Format

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
OV	ME					OV	Ref Cnt					RES					Xfer Register Address					OV	OV	CTX						

Table 3-65. SRAM Journal Indirect Field Definitions

Field	Description
OV [31]	Override bit for Microengine field
ME	For the journal command this specifies the Microengine where the journal data is read from and signaled upon completion. For the fast_journal command, this is the ME number that is written into journal area as part of the tag. [30] = ME Cluster, [29] = RES, [28:26] = ME number. Refer to Section 3.1.2.3.3 .
OV [25]	Override bit for Ref_Cnt field
Ref_Cnt	Overrides the ref_cnt field specified by the instruction. This field supports ref_cnts from 0 to 15 where a value of 0 indicates a transfer of 1 and 15 a transfer of 16. Note that when specifying the Ref_cnt and Xfer Register Address, data can be written into another contexts registers set.
RES	Reserved
Xfer Register Address	Overrides the xfer field specified by the instruction. This field specifies the absolute address of the transfer registers (0-127).
OV [4]	Override bit for Xfer register Address field
OV [3]	Override bit for CTX field
CTX	For the journal command this specifies the context from where the journal data is read from and signaled upon completion. For the fast_journal command, this is the context number that is written into journal area as part of the tag.

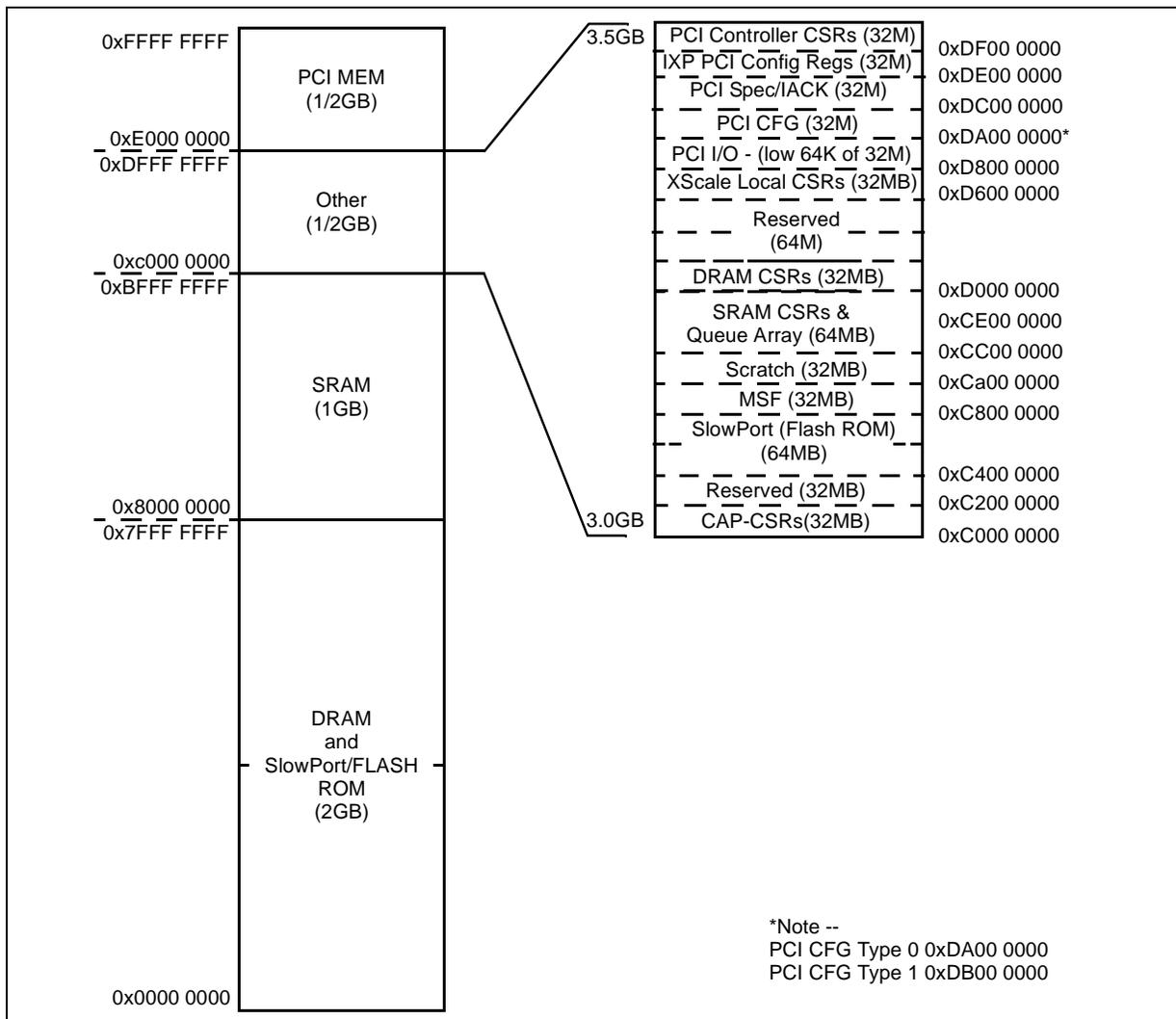
Address Maps

4

4.1 Intel XScale® Address Map

The 4 GB (32 bit address) Intel XScale® address space is divided among the various targets as shown in Figure 4-1. The details of the sub-regions are described in the following subsections. Users should not access address ranges that are not used with respect to the definition in this section.

Figure 4-1. Four GB (32 bit) Intel XScale® Address Space Divided among Various Targets



4.1.1 DRAM Memory and Intel XScale® Core Flash ROM (2GB)

DRAM Memory typically appears in the lower 2GB address space except when the Intel XScale® core is expected to boot the system from Flash ROM. In this case a strap pin aliases the Flash ROM to address 0 and the Intel XScale® software can disable the alias via a CSR bit after it has booted the system. Table 4-1 shows the conditions that cause the Flash ROM to appear at address 0.

Table 4-1. Flash ROM - DRAM Mapping

MISC_CONTROL [Flash Alias Disable] CAP CSR	What is accessed at address...	
	0x0000 0000	0xc4000 0000
0	Flash ROM	Flash ROM
1	DRAM	Flash ROM

In the case of the IXP2800 network processor, the three DRAM channels are interleaved in hardware on 128 byte boundaries to improve concurrency and bandwidth utilization. On the other hand, the IXP2400 network processor supports only one DRAM channel. The mapping of addresses to channels is completely transparent to software, so software deals with physical contiguous addresses in DRAM space. If less than 2 GB of memory is present, the upper part of the address space is not used. Accessing an address above the amount of DRAM populated will cause unpredictable results. Channel interleaving is always performed when either two or three DRAM channels are enabled.

4.1.2 SRAM Memory (1GB)

This address space is used to access SRAM read, write, and the atomic operation functions. The SRAM Queue Array and SRAM CSRs are accessed using SRAM CSR and Queue Array address space (refer to Section 4.1.7).

Table 4-2 shows the details of the SRAM address map. Each SRAM channel is accessed in a different region of the 1GB SRAM Memory address space as shown.

In the case of IXP2400 network processor, only channel 0 and channel 1 are available. The address ranges that channel 2 and channel 3 occupy are Reserved and must not be used. Otherwise, the behavior is undefined.

Each Command (read, write, and the atomic commands) are accessed by the Intel XScale® core using aliases of the address space. For example a write is done to address 0 in Channel 0 at address 0x80000000, while an atomic bit-set is done to address 0 in channel 0 at address 0x84000000.

In order for SRAM atomic and ring commands from Intel XScale® core to be executed properly the page attribute bits have to be set for buffering but no coalescing (xcb = 101).

Software should not generate addresses to SRAM space larger than the amount of SRAM actually present on a channel.

Byte and halfword (2-byte) writes using the Intel XScale® core STRB or STRH instructions result in a masked write operation on the SRAM bus (as opposed to a read-modify-write operation).

Table 4-2. SRAM Address Map for the Intel XScale® Core

Command	Intel XScale® Core Instruction	Channel 0 Base Address	Channel 1 Base Address	Channel 2 Base Address (Reserved on IXP2400)	Channel 3 Base Address (Reserved on IXP2400)
Read	load	0x8000 0000	0x9000 0000	0xA000 0000	0xB000 0000
Write	store				
Swap	swap				
Bit Set	store	0x8400 0000	0x9400 0000	0xA400 0000	0xB400 0000
Bit Test and Set	swap				
Bit Clear	store	0x8800 0000	0x9800 0000	0xA800 0000	0xB800 0000
Bit Test and Clear	swap				
Add	store	0x8C00 0000	0x9C00 0000	0xAC00 0000	0xBC00 0000
Test and Add	swap				
Get (Note 2,3)	load	0xCE00 0000	0xCE40 0000	0xCE80 0000	0xCEC0 0000
Put (Note 2,3)	swap				
Deq (Note 2,4)	load	0xCC00 0100	0xCC40 0100	0xCC80 0100	0xCCC0 0100
Enq (Note 2,4,5)	store				
SRAM csr read	load	0xCC01 0000	0xCC41 0000	0xCC81 0000	0xCCC1 0000
SRAM csr write	store				

NOTE:

- Value is the Base Address of each region. Each region for Read/Write, Bit Set, Bit Clear, Add is 64MB to allow for maximum populated SRAM, however, software must not accesses addresses larger than populated SRAM. Each region for Get/Put and csr read/write is 32 MB. IXP2400 supports channels 0 and 1 only. If addressed ring is configured to return a success/fail code, a swap instruction must be used.
- Each SRAM Controller supports 64 Queue Array Elements that can be used as a Ring or a Queue. The first Queue Array Element is accessed at the base address shown in the table and the other rings are addressed at the next contiguous 4-byte addresses (ring 2 is at base + 4, ring 3 is at base + 8, etc).
- Put operations to Rings use a swap instruction and the data returned to the Intel XScale® core is the status word that indicates whether or not the Put operation was successful. A load multiple instruction can be used to put multiple 4-byte words on a ring, however the swap instruction only support one 4-byte words for Get operations.
- When an Enqueue operation is performed, the data written to memory is a byte address that specifies the address that is placed onto the queue, however the address stored in physical SRAM is a 4-byte addresses (i.e. a byte address shifted right by 2 bits). When a dequeue operation is performed, the 4-byte address is returned to the the Intel XScale® core.
- When performing SRAM Enqueues from the Intel XScale® core, the EOP, SOP, & Segment Count fields are always written with values EOP = 1, SOP = 1, & Segment Count = 0.
- the Intel XScale® core cannot perform enqueue operations if the queue controller is in Mode 3 since a Segment Count of zero on enqueue is illegal in this mode. Refer to [Section 3.2.52](#) for a description of Mode 3.

4.1.3 CAP-CSRs (32MB)

The CSR Access Proxy (CAP) are used to access four memory spaces:

- ME Transfer and Local CSRs via the reflector ([Section 4.1.3.1](#))
- Peripherals via the internal APB bus ([Section 4.1.3.2](#))

- CAP CSRs via internal CAP CSR bus ([Section 4.1.3.3](#))
- SlowPort Interface

Although the SlowPort interface is accessed via CAP it has a separate address range. Refer to [Section 4.1.4](#).

This address space is not byte writeable, and should not be written by the Intel XScale® core with STRB or STRH instructions, otherwise results of the write are unpredictable.

4.1.3.1 ME Transfer and Local CSRs

The Intel XScale® core can:

- Read the S or D write transfer registers of an ME
- Write the S or D read transfer registers of an ME

The Intel XScale® core cannot:

- Read the D write transfer registers of an ME
- Read the S or D read transfer registers of an ME
- Write the S or D write transfer registers of an ME

One address is provided for each read/write transfer register pair so reading an address reads the write transfer register and writing to the same address writes a read transfer registers.

The Intel XScale® core can also read and write the ME's Local CSRs.

Table 4-3. ME Transfer register and Local CSR Address Map for the Intel XScale® Core

ME Register	Base Address	Comments
ME Transfer Registers	0xC000 8000	Where [14]= ME cluster [13]= reserved [12:10]= ME number [9] = S or D Transfer Register [8:2]= Xfer Register address (Refer to Table 4-4).
ME Local CSR	0xC001 8000	Where [14]= ME cluster [13] = reserved [12:10]= ME number [9:0]= CSR address (refer to Table 5-4 for CSRs addresses).

Table 4-4. ME Transfer Register Addresses

Number of Contexts	Context Number	Transfer register
8	0	0-15
	1	16-31
	2	32-47
	3	48-63
	4	64-79
	5	80-95
	6	96-111
	7	112-127

Number of Contexts	Context Number	Transfer register
4	0	0-31
	2	32-63
	4	64-95
	6	96-127

4.1.3.2 Peripherals

The peripherals can be accessed using the memory spaces shown in [Table 4-5](#)

Table 4-5. Peripherals Address Map for the Intel XScale® Core

Peripheral	Address Range	Comments
GPIO	0xC001 0000 - 0xC001 FFFF	Refer to Section 5.6.6 for specific CSR addresses.
Timers	0xC002 0000 - 0xC002 FFFF	Refer to Section 5.6.5 for specific CSR addresses.
UART	0xC003 0000 - 0xC003 FFFF	Refer to Section 5.6.7 for specific CSR addresses.
PMU	0xC005 0000 - 0xC005 FFFF	Refer to Section 5.6.8 for specific CSR addresses.
Slow Port Registers	0xC008 0000 - 0xC008 FFFF	Refer to Section 5.6.9 for specific CSR addresses).

4.1.3.3 CAP CSRs

The Global CSRs can be accessed using the memory spaces shown in [Table 4-6](#)

Table 4-6. CAP CSR Address Map for the Intel XScale® Core

Global CSR	Base Address	Comments
Global	0xC000 4A00	Miscellaneous system level GPRs. Refer to Section 5.6.4 for specific CSR addresses).
Hash Configuration	0xC000 4900	Refer to Section 5.6.2 for specific CSR addresses.
Scratchpad Configuration	0xC000 4800	Refer to Section 5.6.1 for specific CSR addresses.
“Fast Write” CSRs	0xC000 4000	The Intel XScale® core can read and write these registers but the fast write capability is not supported. Refer to Section 5.6.3 for specific CSR addresses.

4.1.4 SlowPort - Flash ROM (64M)

The total address space is 64MB, which is further divided into two 32MB segments. Each segment generates a SlowPort Chip select signal. If the devices on the SlowPort have a density of 256Mbit (32MB) each, all the address space is going to be filled like a contiguous address space. However, if a small capacity device is used, like 4MB, 8MB, 16MB, there will be a memory ‘hole’ left in between these two devices. The holes should not be accessed.

Table 4-7. Slow Port Address Map for the Intel XScale® Core

Global CSR	Address Range	Comments
Slow Port Device 0	0xC400 0000 - 0xC5FF FFFF	This 32M address range asserts the chip select signal SP_CS_L[0] provided by the IXP2800. This space is typically used for Flash ROM.
Slow Port Device 1	0xC600 0000 - 0xC7FF FFFF	This 32M address range asserts the chip select signal SP_CS_L[1] provided by the IXP2800. This space is typically used for Flash ROM or control port for Media devices.

4.1.5 MSF (32M)

The Media Switch Fabric (MSF) Unit can be accessed using the memory space 0xC800 0000 to 0xC800 FFFF. The Intel XScale® core can perform read and write operations to MSF, but it can not perform fast write operations

This address space is not byte writeable, and should be not be written by the Intel XScale® core with STRB or STRH instructions, otherwise results of the write are unpredictable.

Table 4-8. MSF Address Map for the Intel XScale® Core

Global CSR	Address Range	Comments
MSF CSRs	0xC800 0000 - 0xC800 1FFF	Refer to Section 5.7 and Section 5.8 .
RBUF	0xC800 2000 - 0xC800 3FFF	Read only. The RBUF can not be written by the Intel XScale® core.
TBUF	0xC800 2000 - 0xC800 3FFF	Write only. The TBUF can not be read by the Intel XScale® core
IXP2800 RCOMP CSRs	0xC800 8008 - 0xC800 800C	Refer to Section 5.7 .

The 8K RBUF and TBUF can be configured to support different size elements which effects the number of elements in the RBUF and TBUF. The following tables show how the addresses can be interpreted as element numbers an offsets into the elements. Bits [12:0] can be viewed as a byte address and the ignored field represent the byte offset which is ignored by the MSF.

Table 4-9. RBUF/ TBUF Offset Address 128 64-Byte Elements

1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
5	4	3	2	1	0											
0	0	1	Element number								4-byte aligned offset into element				ignored	

Table 4-10. RBUF/ TBUF Offset Address 64 128-Byte Elements

1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
5	4	3	2	1	0											
0	0	1	Element number								4-byte aligned offset into element				Don't care	

Table 4-11. RBUF/ TBUF Offset Address 32 256-Byte Elements

1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
5	4	3	2	1	0											
0	0	1	Element number								4-byte aligned offset into element				Don't care	

4.1.6 Scratch (32M)

Table 4-12 shows the details of the Scratch address map. The Intel XScale® core maps various read-modify-write operations to Scratch memory using alias addresses.

The scratch rings are configured via CSRs that reside in the CAP memory space (Refer to Section 4.1.3). The Scratch Ring Full signals can be accessed via the Intel XScale® core Interrupt Controller which is accessed using the Intel XScale® core Local CSRs (Refer to Section 4.1.9)

In order for Scratch atomic and ring commands from Intel XScale® core to be executed properly the page attribute bits have to be set for buffering but no coalescing (xcb = 101).

Scratch memory is not byte writeable, and should be not be written by Intel XScale® core with STRB or STRH instructions, otherwise results of the write are unpredictable.

Table 4-12. Scratch Address Map

Command	Instruction	Address Range
Read	load	0xCA00 3FFF - 0xCA00 0000
Write	store	
Swap	swap	
Bit Set	store	0xCA40 3FFF - 0xCA40 0000
Bit Test and Set	swap	
Bit Clear	store	0xCA80 3FFF - 0xCA80 0000
Bit Test and Clear	swap	
Add	store	0xCAC0 3FFF - 0xCAC0 0000
Test and Add	swap	
Subtract	store	0xCB00 3FFF - 0xCB00 0000
Test and Subtract	swap	
Get	load	There are 16 rings and the Ring Number is specified in bits [5:2] of the base address 0xCB40 00xx. Therefore “xx” in the base address for the 16 rings are as follows: Rings 0 - 0x00 Rings 8 - 0x20 Rings 1 - 0x04 Rings 9 - 0x24 Rings 2 - 0x08 Rings 10 - 0x28 Rings 3 - 0x0C Rings 11 - 0x2C Rings 4 - 0x10 Rings 12 - 0x30 Rings 5 - 0x14 Rings 13 - 0x34 Rings 6 - 0x18 Rings 14 - 0x38 Rings 7 - 0x1C Rings 15 - 0x3C
Put	store	

4.1.7 SRAM CSRs and Queue Array (64MB)

The Intel XScale® core can access the SRAM CSRs as well as some of the functionality of the SRAM controller’s SRAM Queue Array. [Table 4-13](#) shows the address mapping.

This address space is not byte writeable, and should be not be written by Intel XScale® core with STRB or STRH instructions, otherwise results of the write are unpredictable.

Table 4-13. SRAM Queue Array Address for the Intel XScale® Core

Operation	Command	Intel XScale® Core Instruction	Base address	Comments
Enqueue	enqueue	Store	CH0: 0xCC00 0100 CH1: 0xCC40 0100 CH2: 0xCC80 0100 CH3: 0xCCC0 0100 (CH2 & CH3 are Reserved on IXP2400)	Each address space supports 64 4-byte word addresses, one for each SRAM Queue Array Entry. The Queue Array Entries (0 to 63) are selected using bits[7:2]
	enqueue_tail	not supported		
Dequeue	dequeue	Load		
Read Descriptor	rd_qdesc_head	not supported		
	rd_qdesc_tail			
	rd_qdesc_other			
Write Descriptor	Write_desc	not supported		
	wr_qdesc_count			
Ring	get	Load	CH0: 0xCE00 0000 CH1: 0xCE40 0000 CH2: 0xCE80 0000 CH3: 0xCEC0 0000 (CH2 & CH3 are Reserved on IXP2400)	Note 1: The put operation requires a swap instruction and can only be used to get one 4-byte word from the ring. The put data is placed onto the ring and the swap data is the status word indicating whether or not the data was placed onto the Ring. The Queue Array Entries (0 to 63) are selected using bits[7:2]
	put	swap (Note 1)		
Journal	jour	not supported		
csr read	csr_rd	Load	CH0: 0xCC01 0000 CH1: 0xCC41 0000 CH2: 0xCC81 0000 CH3: 0xCCC1 0000 (CH2 & CH3 are Reserved on IXP2400)	Refer to Section 5.6 for CSR offsets and descriptions.
csr write	csr_wr	Store		

4.1.8 DRAM CSRs (32M)

The CSRs for each of the DRAM controllers can be accessed using the memory spaces shown in [Table 4-14](#). This address space is not byte writeable, and should be not be written by Intel XScale® core with STRB or STRH instructions, otherwise results of the write are unpredictable.

Table 4-14. DRAM CSRs

DRAM Channel	Base address	Comments
Channel 0	0xD000 9000	Refer to Section 5.3 for CSR offsets and descriptions. Channel 1 and Channel 2 are for IXP2800 only.
Channel 1	0xD000 A000	
Channel 2	0xD000 B000	

4.1.9 Intel XScale® Core Local CSRs (32M)

These registers are local to the Intel XScale® core and can only be accessed by the Intel XScale® core. This address space is not byte writeable, and should not be written by Intel XScale® core with STRB or STRH instructions, otherwise results of the write are unpredictable.

Table 4-15. Intel XScale® Core CSRs

SRAM Channel	Base address	Comments
Interrupt Controller	0xD6FF FFFF - 0xD600 0000	Refer to Section 5.10.1 for CSR offsets and descriptions.
Hash Operations	0xD7FF FFFF - 0xD700 0000	Refer to Section 4.1.9.1 for description on how to access. Refer to Section 5.6.2 for CSR offsets and descriptions.
Break Point		Refer to Section 5.10.3 for CSR offsets and descriptions.

4.1.9.1 Hash Operations

The Intel XScale® core initiates a hash operation by writing a set of memory-mapped Hash Data Registers with the data to be used to generate the hash index. There are a separate set of registers for 48-bit, 64-bit, and 128-bit hash registers, as shown in the [Table 4-16](#). Only one hash operation of each type can be performed at a time. Data write order is 3-2-1-0(for hash_128), 1-0(for hash_48 or hash_64). Writing to register 0 in a set initiates the write operation to the Hash Unit.

The Intel XScale® core reads the results from the Hash Data Registers. Because of queuing delays at the Hash Unit, the time to complete an operation is not fixed. The Intel XScale® core polls the Hash Done Register to determine when the hash result has been returned. This register is cleared when the Hash Data Registers are written to the Hash Unit. Bit [0] of Hash Done Register is set when the Hash Data Registers get the return result from the Hash Unit (when the last word of the result is returned). The Intel XScale® software can poll on Hash Done, and read Hash Result when Hash Done is equal to 0x00000001

Table 4-16. Intel XScale® Core Hash Operand and Results Registers

Operation		Register Name	Register Address
48-Bit Hash			
hash [31:0]		HASH_OP_48_0	0xD700 0000
Don't care	hash [47:32]	HASH_OP_48_1	0xD700 0004
64-Bit Hash			
hash [31:0]		HASH_OP_64_0	0xD700 0010
hash [63:32]		HASH_OP_64_1	0xD700 0014
128-Bit Hash			
hash [31:0]		HASH_OP_128_0	0xD700 0020
hash [63:32]		HASH_OP_128_1	0xD700 0024
hash [64:95]		HASH_OP_128_2	0xD700 0028

Table 4-16. Intel XScale® Core Hash Operand and Results Registers

Operation	Register Name	Register Address
hash [127:96]	HASH_OP_128_3	0xD700 002C
Poll for hash result	HASH_DONE	0xD700 0030
1. Data written to the hash register addresses will be hashed 2. Data read from the hash register addresses will be the hash result		

4.1.10 PCI IO (32M)

Reading or writing this memory space will cause a read or a write on the PCI bus using the PCI I/O commands. Although the memory block is 32M, only the lowest 64K generates I/O cycles.

Table 4-17. PCI I/O Space

Command Type	Address Range
PCI I/O	0xD900 FFFF - 0xD800 0000

For IXP2400 and IXP2800 rev A, PCI read transactions which are originated from the integrated Intel XScale® core are always 32-bit. In other words, when the integrated the Intel XScale® core generates a read transaction that accesses less than 32-bit of data in the PCI address space, the read transaction on the PCI bus will be a 32-bit read transaction aligned to the 32-bit aligned address. PCI write transactions or PCI transactions that originate from micro-engines are not subject to this restriction.

4.1.11 PCI CFG (32M)

PCI supports two types of configuration transactions: Type 0 and Type 1. The IXP2800 and IXP2400 support both types using two memory spaces. Reading or writing these memory spaces will cause a read or a write on the PCI bus using the PCI configuration command.

Table 4-18. PCI Configuration Space

Command Type	Address Range
PCI Configuration Type 0	0xDAFF FFFF - 0xDA000000
PCI Configuration Type 1	0xDBFF FFFF - 0xDB000000

4.1.12 PCI Special Cycles / IACK (32M)

Reading or writing this memory space will cause a read or a write on the PCI bus using the PCI Special or IACK commands.

Table 4-19. PCI Configuration Space

Command Type	Address Range	Comments
PCI Special Cycles	0xDDFF FFFF - 0xDC00 0000	Reading this memory space will result in an IACK command on the PCI bus.
PCI IACK		Writing this memory space will result in an Special Cycle command on the PCI bus.

4.1.13 PCI Configuration Registers (32M)

The Intel XScale® core can access the IXP2400/IXP2800 PCI Configuration registers using this memory space. Access to this space is local to the IXP2400/IXP2800 and does not result in a PCI bus transaction. This address space is not byte writeable, and should be not be written by the Intel XScale® core with STRB or STRH instructions, otherwise results of the write are unpredictable.

Table 4-20. IXP2400/IXP2800 PCI Configuration Space

Command Type	Address Range	Comments
IXP2400/IXP2800 PCI Configuration registers	0xDEFF FFFF - 0xDE000000	Refer to Section 5.9.1 for CSR offsets and descriptions.

4.1.14 PCI Controller CSRs

The Intel XScale® core can access the IXP2400/IXP2800 PCI Controllers CSRs using this memory space. Access to this space is local to the IXP2400/IXP2800 and does not result in a PCI bus transaction. This address space is not byte writeable, and should be not be written by the Intel XScale® core with STRB or STRH instructions, otherwise results of the write are unpredictable.

Table 4-21. IXP2400/IXP2800 PCI Controller CSR Space

Command Type	Address Range	Comments
IXP2400/IXP2800 PCI Controller CSRs	0xDFFF FFFF - 0xDF000000	Refer to Section 5.9.2 for register descriptions and offsets

4.1.15 PCI Memory (1/2GB)

Reading or writing this memory space will cause a read or a write on the PCI bus using the PCI Memory command.

For IXP2400 and IXP2800 rev A, PCI read transactions which are originated from the integrated Intel XScale® core are always 32-bit. In other words, when the integrated Intel XScale® core generates a read transaction that accesses less than 32-bit of data in the PCI address space, the read transaction on the PCI bus will be a 32-bit read transaction aligned to the 32-bit aligned address. PCI write transactions or PCI transactions that originate from micro-engines are not subject to this restriction.

Table 4-22. IXP2400/IXP2800 PCI Configuration Space

Command Type	Address Range	Comments
PCI Memory Space	0xFFFF FFFF - 0xE000 0000	

4.2 PCI Address Map

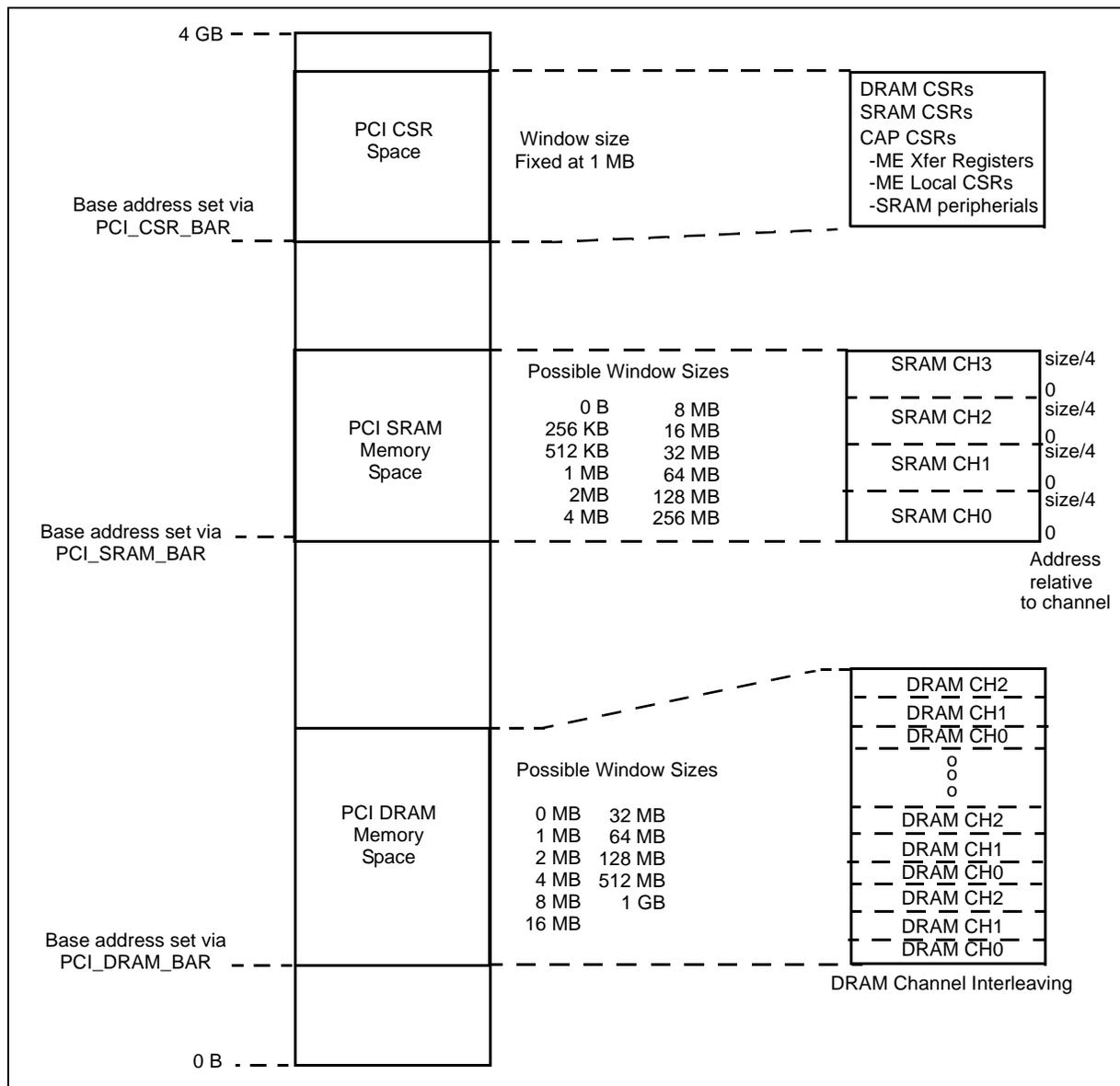
The IXP2400/IXP2800 can be mapped into three different memory spaces within the 4 GB (32 bit address) PCI address space. These three spaces are configured via Base Address Registers (BAR) visible via the PCI configuration space.

If the system is booted using the Intel XScale® processor, the BAR registers are configured by the Intel XScale® core writing to the BAR.

In the case of the IXP2400 network processor, there are only 2 SRAM channels. Nevertheless, the SRAM BAR configuration still assumes 4 channels. The address ranges that are mapped to channel 2 and channel 3 are Reserved. Transactions must not target these address ranges. Otherwise, the behavior is undefined.

In the case of IXP2400 network processor, there is only 1 DRAM channel. As a result, there is no channel interleave effect, and the whole DRAM address range is mapped onto the single channel.

Figure 4-2. Four GB (32 bit) PCI Address Space



4.2.1 DRAM Memory Space

If the Intel XScale® core boots from the system, the DRAM_BASE_ADDR_MASK register determines the window size. If the system boots from PCI, the PCI_SWIN strap pins determine the window size. The window always starts at DRAM address 0x0000 0000. The DRAM CSRs are accessed using PCI CSR Memory space (refer to Section 4.2.3).

4.2.2 SRAM Memory Space

If the Intel XScale® core boots from the system, the SRAM_BASE_ADDR_MASK register determines the window size. If the system boots from PCI, the PCI_SWIN strap pins determine the window size.

PCI only has access to the SRAM read and write operations; the SRAM Queue Array, and Atomic operations are not supported for PCI. The SRAM CSRs are accessed using PCI CSR Memory space (refer to [Section 4.2.3](#)).

The four SRAM Controllers are divided into the window size such that each controller gets an equal share of the window space. The window always starts at the low addresses in the SRAM Controller.

In the case of the IXP2400 network processor, the address ranges that map to Controllers 2 and 3 are Reserved.

Table 4-23. PCI Address Offset vs SRAM Controller

Window Size in SRAM BAR	PCI Address Offset				Total Size for each Memory Controller starting at address 0 relative to the SRAM Controller
	Controller 0	Controller 1	Controller 2 (Reserved on IXP2400)	Controller 3 (Reserved on IXP2400)	
0	SRAM Memory is not visible to PCI				
256K	0x0000 FFFF-0x0000 0000	0x0001 FFFF-0x0001 0000	0x0002 FFFF-0x0002 0000	0x0003 FFFF-0x0003 0000	64K
512K	0x0001 FFFF-0x0000 0000	0x0003 FFFF-0x0002 0000	0x0005 FFFF-0x0004 0000	0x0007 FFFF-0x0006 0000	128K
1MB	0x0003 FFFF-0x0000 0000	0x0007 FFFF-0x0004 0000	0x000B FFFF-0x0008 0000	0x000F FFFF-0x000C 0000	256K
2MB	0x0007 FFFF-0x0000 0000	0x000F FFFF-0x0008 0000	0x0017 FFFF-0x0010 0000	0x001F FFFF-0x0018 0000	512K
4MB	0x000F FFFF-0x0000 0000	0x001F FFFF-0x0010 0000	0x002F FFFF-0x0020 0000	0x003F FFFF-0x0030 0000	1MB
8MB	0x001F FFFF-0x0000 0000	0x003F FFFF-0x0020 0000	0x005F FFFF-0x0040 0000	0x007F FFFF-0x0060 0000	2MB
16MB	0x003F FFFF-0x0000 0000	0x007F FFFF-0x0040 0000	0x00BF FFFF-0x0080 0000	0x00FF FFFF-0x00C0 0000	4MB
32MB	0x007F FFFF-0x0000 0000	0x00FF FFFF-0x0080 0000	0x017F FFFF-0x0100 0000	0x01FF FFFF-0x0180 0000	8MB
64MB	0x00FF FFFF-0x0000 0000	0x01FF FFFF-0x0100 0000	0x02FF FFFF-0x0200 0000	0x03FF FFFF-0x0300 0000	16MB
128MB	0x01FF FFFF-0x0000 0000	0x03FF FFFF-0x0200 0000	0x05FF FFFF-0x0400 0000	0x07FF FFFF-0x0600 0000	32MB
256MB	0x03FF FFFF-0x0000 0000	0x07FF FFFF-0x0400 0000	0xBFFF FFFF-0x8000 0000	0xFFFF FFFF-0xC000 0000	64MB

4.2.3 CSR Memory Space

The PCI CSR memory space has a fixed window size of 1MB. [Table 4-24](#) shows what can be accessed using this address space.

Table 4-24. CSR Memory Space for PCI

Global CSR	Address Range	Size Allocated	Comments
PCI Controller CSR	0xF FFFF - 0xF E000	8KB	Refer to Section 5.9.2 for register descriptions and offsets
Reserved	0xF DFFF - 0xF D800	2KB	
DRAM CSR CH2	0xF D7FF - 0xF D000	2KB	Refer to Section 5.4.2 for CSR offsets and descriptions. CH1 and CH2 are Reserved on IXP2400.
DRAM CSR CH1	0xF CFFF - 0xF C800	2KB	
DRAM CSR CH0	0xF C7FF - 0xF C000	2KB	
Reserved	0xF BFFF - 0xF A000	8KB	
SRAM CSR CH3	0xF 9FFF - 0x F9C00	1KB	Refer to Section 5.5.1 for CSR offsets and descriptions. CH2 and CH3 are Reserved on IXP2400.
SRAM CSR CH2	0xF 9BFF - 0xF 9800	1KB	
SRAM CSR CH1	0xF 97FF - 0xF 9400	1KB	
SRAM CSR CH0	0xF 93FF - 0xF 9000	1KB	
Reserved	0xF 8FFF - 0xF 8000	4KB	
MSF CSRs	0xF 5FFF - 0xF 4000	8KB	Refer to Section 5.7 .
MSF RBUF	0xF 7FFF - 0xF 6000	8KB	Read only
MSF TBUF			Write only
Scratch Memory	0xF 3FFF - 0xF 0000	16KB	
CAP CSRs	0xE FFFF - 0x0 0000	983 KByte	See Table 4-25 for a breakdown of this address space.

Table 4-25. CAP CSR Memory Space Breakdown for PCI

Global CSR	Base Address	Size Allocated	Comments
ME Transfer Registers	0x0 8000	983 KByte	<p>Where [14]= ME cluster [13]= reserved [12:10]= ME number [9] = S or D Transfer Register [8:2]= Xfer Register address (Refer to Table 4-4).</p> <p>The PCI Host can read the write transfer register of an ME and it can write the read transfer registers of an ME. It is not possible to write the write transfer registers or read the read transfer registers. Reading an address reads the write transfer register and writing to the same address writes a read transfer registers. It is not possible to write the write transfer registers or read the read transfer registers.</p>
ME Local CSR	0x1 8000		<p>Where [14]= ME cluster [13]= reserved [12:10]= ME number [9:0]= CSR address (refer to Table 5.2 for CSR addresses).</p>
SlowPort CS0	not supported		This 32M address range asserts the chip select signal SP_CS_L[0] provided by the IXP2800. This space is typically used for Flash ROM.
SlowPort CS1	not supported		This 32M address range asserts the chip select signal SP_CS_L[1] provided by the IXP2800. This space is typically used for Flash ROM or control port for Media devices.
GPIO	0x1 0000		Refer to Section 5.6.6 for specific CSR addresses.
Timers	0x2 0000		Refer to Section 5.6.5 for specific CSR addresses.
UART	0x3 0000		Refer to Section 5.6.7 for specific CSR addresses.
PMU	0x5 0000		Refer to Section 5.6.8
Slow Port Registers	0x8 0000		Refer to Section 5.6.9 for specific CSRs addresses.
Global	0X0 4A00		Miscellaneous system level GPRs. Refer to Section 5.6.4 for specific CSRs addresses).
Hash Configuration	0X0 4900		Refer to Section 5.6.2 for specific CSR addresses.
Scratch Configuration	0X0 4800		Refer to Section 5.6.1 for specific CSR addresses.
Fast Write CSRs	0X0 4000		The PCI Host can read and write these registers but the fast write capability is not supported. Refer to Section 5.6.3 for specific CSR addresses.

4.3 Microengine Address Map

The MEs support command that allow access to various functionality listed in [Figure 4-26](#).

Table 4-26. ME I/O Access

Unit	Functionality	Instruction/ Commands	Comments
CAP	Intel XScale® UART Intel XScale® Timers Intel XScale® GPIO Intel XScale® SlowPort Interface IXP Global CSRs Scratch CSRs Hash CSRs PMU CSRs Slow Port CSRs Slow Port Memory	CAP[read] CAP[write]	The calculated addressing version of the instruction must be used. Refer to Section 3.2.19 for addresses.
	Fast Write CSRs	CAP[fast_wr] CAP[read] CAP[write]	Either the enumerated CSR addressing or calculated addressing version of the instruction can be used for read and write. Refer to Section 3.2.18 and Section 3.2.19 for addresses.
ME	MEs Own Local CSRs	LOCAL_CSR_RD LOCAL_CSR_WR	CSR names are used to specify the CSRs. Refer to the LOCAL_CSR_RD and LOCAL_CSR_WR instructions for a list of supported CSR names.
	Other ME Transfer Registers	CAP[] (Reflect and Calculated addressing versions)	When the Reflect version is used, Transfer Register names are supported (Refer to Section 3.2.20). Refer to Section 3.2.19 for Calculated addresses.
	Other ME Local CSRs	CAP[] (Calculated addressing version)	Refer to Section 3.2.19 for addresses.
	Other ME GPRs	No Access	
MSF	RBUF (read only)	MSF[read] MSF[read64]	The RBUF is accessed using a read or read64 command and base address of 0x2000.
	TBUF (write only)	MSF[write] MSF[write64]	The TBUF is accessed using a write or write64 command and base address of 0x2000.
	MSF CSRs	MSF[read] MSF[read] MSF[fast_wr]	MSF CSRs are accessed using addresses between 0 and 0x1FFF. Refer to the MSF instruction for a list of supported CSRs and their addresses.

Table 4-26. ME I/O Access

Unit	Functionality	Instruction/ Commands	Comments
PCI	PCI Memory Space	PCI[read] PCI[write]	Use address range: 0x2000 0000 to 0x3FFF FFFF
	PCI I/O		Use address range: 0x0000 0000 to 0x0000 FFFF
	PCI CFG Type 0		Use address range: 0x0200 0000 to 0x02FF FFFF
	PCI CFG Type 1		Use address range: 0x0300 0000 to 0x03FF FFFF
	PCI Controller CSRs		Use address range: 0x0600 0000 to 0x07FF FFFF
	PCI Special	PCI[write]	Use address range: 0x0400 0000 to 0x05FF FFFF
	PCI IACK	PCI[read]	
Scratch Memory		scratch[read] scratch[write] scratch[swap] scratch[set] scratch[clr] scratch[incr] scratch[decr] scratch[add] scratch[sub] scratch[test_and_set] scratch[test_and_clr] scratch[test_and_incr] scratch[test_and_decr] scratch[test_and_add] scratch[test_and_sub] scratch[get] scratch[put]	Use address range: 0x0000 0000 to 0x0000 3FFF

Table 4-26. ME I/O Access

Unit	Functionality	Instruction/ Commands	Comments
SRAM	SRAM CSRs	sram[csr_rd] sram[csr_wr]	
	SRAM Memory	sram[read] sram[write] sram[swap] sram[set] sram[clr] sram[incr] sram[decr] sram[add] sram[test_and_set] sram[test_and_clr] sram[test_and_incr] sram[test_and_decr] sram[test_and_add]	The base addresses of the SRAM channels are: Channel 0: 0x0000 0000 Channel 1: 0x4000 0000 Channel 2: 0x8000 0000 Channel 3: 0xc000 0000 Channels 2 & 3 are Reserved on IXP2400. The size of the memory installed determines the upper bound of the address.
	SRAM Queue Array	sram[rd_qdesc_head] sram[rd_qdesc_tail] sram[rd_qdesc_other] sram[wr_qdesc] sram[wr_qdesc_count] sram[enqueue] sram[enqueue_tail] sram[dequeue] sram[get] sram[put] sram[journal] sram[fast_journal]	The Queue Array entry and SRAM Channel and SRAM address is determined by src_opA + src_opB. Refer to the instruction definition for more details.
DRAM	CSRs	No Access	
	DRAM Memory	DRAM[read] DRAM[write]	DRAM begins at address 0x0000 0000 and the size of the memory installed determines the upper bound of the address. For the IXP2800, the three channels are interleaved.
HASH	Hash operations	hash_48[] hash_64[] hash_128[]	
Intel XScale® Core Local CSRs	Interrupt Controller Scratch Ring Full Status Intel XScale® core Hash Result Register	No Access	
Note 1: The address of the registers is identical to that used by PCI as shown in Table 4-25 without the base address shown in Table 4-24			

Control and Status Registers (CSRs) 5

5.1 Introduction

This chapter describes the internal registers of the IXP2800 and IXP2400. In addition, there is a section that summarizes the differences between IXP2400 and IXP2800 in terms of configuring and programming the Media and Switch Fabric Interface unit (MSF).

5.1.1 IXP2800 and IXP2400 CSR Summary

The registers in this chapter are divided into the sections based on the functional unit in which the register resided. The CSRs can be found in the sections as shown in [Table 5-1](#).

Table 5-1. CSR Summary

Unit	CSRs	Section
Microengine	Local CSRs	Section 5.2
RDR DRAM (IXP2800)	IXP2800 RDR DRAM CSRs	Section 5.3
DDR DRAM (IXP2400)	IXP2400 DDR DRAM CSRs	Section 5.4
SRAM	SRAM CSRs	Section 5.5
CSR Access Proxy (CAP)	Scratch CSRs	Section 5.6.1
	Hash CSRs	Section 5.6.2
	Intel XScale® Core Timers	Section 5.6.5
	Intel XScale® Core GPIO	Section 5.6.6
	Intel XScale® Core UART	Section 5.6.7
	PMU (Performance Monitor Unit)	Section 5.6.8
	Intel XScale® Core SlowPort Interface	Section 5.6.9
	Fast Write CSRs	Section 5.6.3
	IXP Global CSRs	Section 5.6.4
MSF (IXP2800)	RBUF TBUF IXP2800 MSF CSRs	Section 5.7
MSF (IXP2400)	RBUF TBUF IXP2400 MSF CSRs	Section 5.8
PCI	PCI Configuration	Section 5.9.1
	PCI Controller	Section 5.9.2

Table 5-1. CSR Summary

Unit	CSRs	Section
Intel XScale™ Core Local CSRs	Interrupt Controller and Scratch Ring Full Status	Section 5.10.1
	Intel XScale™ core Hash Result Register	Section 5.10.2
	Breakpoint	Section 5.10.3
Intel XScale™ core Co-Processor	Co-Processor 14 Co-Processor 15	Section 5.11

5.1.2 Register Notation Conventions

The register descriptions that follow use the following notation conventions:

Table 5-2. Register Notation Conventions

RW Field	
Notation	Description
RO	Read only. Write may result in unpredictable behavior.
RW	Read and Write
RW1C	Read and Write 1 to Clear
RC	Clear on Read
WO	Write only. Read may result in unpredictable behavior.
W1C	Write 1 to clear
W1S	Write 1 to set
WRC	Write and Clear on Read
Reset Field	
Notation	Description
undef	Undefined. For read, indeterminate data is returned.
0x	Specifies that the number is in hexadecimal format.
dep	Dependent on an external source such as a external pin state. Refer to the description for details.
--	In context of the RW field, Write data is ignored.

5.1.3 Reserved Fields

All the reserved bits are read zero and have no effect for write access. Since all the reserved bits are read zero, the individual register descriptions state that the reset value of reserved bits are “0”.

Intel reserves the right to change the usage of these bits in the future. When new functionality is added, and when possible, Intel will require a “1” be written to enable the new functionality.

To increase the likelihood of software backward compatibility, it is highly recommended that software treat reserved bits as undefined during read operations (that is, software should read the register and mask all reserved bits to zero), and always write “0” to reserved bits during write operations.

Software must not access register locations that are not defined in this document. Such accesses may result in unpredictable behavior.

5.2 Microengine Local CSRs

Table 5-3 shows the offset addresses of the ME Local CSRs. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can be accessed from the Intel XScale® core, PCI, and a remote MEs via CAP. An ME can access its own Local CSRs using the local_csr_write and local_csr_read instructions.

Some local CSRs listed in the table are actually 8 register: one for each ME context. These registers are accessed using a common address and an indirection pointer that specifies the context. The indirect pointer is set via the CSR_CTX_POINTER Local Registers (Section 5.2.20). When a context starts executing, the per-context-register is copied into a working (or active) register and when it stops executing (swaps out) the working register is saved back into the per-context-register.

Register fields that are not cleared by hardware reset are listed as “Undef”. This specifically means that they are not changed by reset, and can be read for debug after the Microengine has been reset.

Table 5-3. Microengine Local CSR Summary (Sheet 1 of 3)

Register Name	CSR Addr offset	Comment	Section
USTORE_ADDRESS	0x00	These addresses only apply to external accesses (from the Intel XScale® core). These registers are not accessible by local_csr_rd and local_csr_wr.	Section 5.2.1
USTORE_DATA_LOWER	0x04		Section 5.2.2
USTORE_DATA_UPPER	0x08		Section 5.2.2
USTORE_ERROR_STATUS	0x0C		Section 5.2.3
ALU_OUT	0x10		Section 5.2.4
CTX_ARB_CNTL	0x14		Section 5.2.17
CTX_ENABLES	0x18		Section 5.2.18
CC_ENABLE	0x1C		Section 5.2.19
CSR_CTX_POINTER	0x20		Section 5.2.20
INDIRECT_CTX_STS	0x40		Section 5.2.16
ACTIVE_CTX_STS	0x44		Section 5.2.15

Table 5-3. Microengine Local CSR Summary (Sheet 2 of 3)

Register Name	CSR Addr offset	Comment	Section
INDIRECT_CTX_SIG_EVENTS	0x48	Indirect address accesses register selected by the value in the CSR_CTX_POINTER[2:0]. Active address accesses the register selected by ACTIVE_CTX_STS[2:0].	Section 5.2.22
ACTIVE_CTX_SIG_EVENTS	0x4C		Section 5.2.21
INDIRECT_CTX_WAKEUP_EVENTS	0x50		Section 5.2.24
ACTIVE_CTX_WAKEUP_EVENTS	0x54		Section 5.2.23
INDIRECT_CTX_FUTURE_COUNT	0x58		Section 5.2.7
ACTIVE_CTX_FUTURE_COUNT	0x5C		Section 5.2.6
INDIRECT_LM_ADDR_0,	0x60	Access LM_ADDR_0 selected by the value in the CSR_CTX_Pointer[2:0].	Section 5.2.27
ACTIVE_LM_ADDR_0,	0x64	Access working copy LM_ADDR_0.	Section 5.2.25
INDIRECT_LM_ADDR_1	0x68	Access LM_ADDR_1 selected by the value in the CSR_CTX_Pointer[2:0].	Section 5.2.28
ACTIVE_LM_ADDR_1	0x6C	Access working copy LM_ADDR_1.	Section 5.2.26
BYTE_INDEX	0x70	Access BYTE_INDEX.	Section 5.2.29
ACTIVE_LM_ADDR_0_BYTE_INDEX	0xE4	Access LM_ADDR_0_INDIRECT[11:2] concatenated to BYTE_INDEX[1:0].	Section 5.2.34
INDIRECT_LM_ADDR_0_BYTE_INDEX	0xE0	Access working copy LM_ADDR_0[11:2] concatenated to BYTE_INDEX[1:0].	Section 5.2.32
INDIRECT_LM_ADDR_1_BYTE_INDEX	0xE8	Access LM_ADDR_1_INDIRECT[11:2] concatenated to BYTE_INDEX[1:0].	Section 5.2.33
ACTIVE_LM_ADDR_1_BYTE_INDEX	0xEC	Access working copy LM_ADDR_1[11:2] concatenated to BYTE_INDEX[1:0].	Section 5.2.35
T_INDEX_BYTE_INDEX	0xF4	Access T_INDEX[8:2] concatenated to BYTE_INDEX[1:0].	Section 5.2.31
T_INDEX	0x74		Section 5.2.30
INDIRECT_FUTURE_COUNT_SIGNAL	0x78	Indirect address accesses register selected by the value in the CSR_CTX_POINTER[2:0].	Section 5.2.9
ACTIVE_FUTURE_COUNT_SIGNAL	0x7C		Section 5.2.8
NN_PUT	0x80		Section 5.2.36
NN_GET	0x84		Section 5.2.37
TIMESTAMP_LOW	0xC0		Section 5.2.5
TIMESTAMP_HIGH	0xC4		Section 5.2.5
RESERVED	0xC8		
NEXT_NEIGHBOR_SIGNAL	0x100		Section 5.2.12
PREV_NEIGHBOR_SIGNAL	0x104		Section 5.2.13
SAME_ME_SIGNAL	0x108		Section 5.2.14
CRC_REMAINDER	0x140		Section 5.2.38
PROFILE_COUNT	0x144		Section 5.2.10
PSEUDO_RANDOM_NUMBER	0x148		Section 5.2.11

Table 5-3. Microengine Local CSR Summary (Sheet 3 of 3)

Register Name	CSR Addr offset	Comment	Section
RESERVED	0x14C to 0x17C		
LOCAL_CSR_STATUS	0x180	This address only applies to external reads (from the Intel XScale® core). This register is not accessible by local_csr_rd. It is a read-only register, so local_csr_wr does not apply.	Section 5.2.39
RESERVED	0x3FC	This address is reserved. No register is accessed. This address can always be used and no register will be written. It can be used to deliver an Event Signal on without changing any registers.	

There are three latencies associated with a local_CSR_Write operations and these latencies and they vary depending on the local CSR as shown in [Table 5-4](#).

- Write Latency: The number of instructions from local_csr_wr to when the Local CSR is actually written.
- Read Latency: The number of instructions between a local_csr_wr to a local_csr_rd of the same register, in order to get the newly written value.
- Usage Latency: The number of instructions between a local_csr_wr to when a function is performed by the hardware.

Table 5-4. Microengine Local CSR Latencies (Sheet 1 of 2)

Register Name	Latency			Usage Latency Comments
	Write	Read	Usage	
USTORE_ADDRESS	NA	NA	NA	
USTORE_DATA_LOWER	NA	NA	NA	
USTORE_DATA_UPPER	NA	NA	NA	
USTORE_ERROR_STATUS	NA	NA	NA	
ALU_OUT	NA	NA	NA	
CTX_ARB_CNTL	4	3	NA	
CTX_ENABLES	4	3	8	A ctx_arb instruction should be issued 8 cycles after an instruction is issued to clear the context-enable bits to ensure that the disabled contexts will not run. NOTE -- This latency may change in future versions of the ME.
CC_ENABLE	4	3	1	Must wait 1 cycle before issuing instruction that modifies CC's
CSR_CTX_POINTER	3	2	3	Must wait 3 cycles before issuing instruction that uses CSR_CTX_Pointer
INDIRECT_CTX_STS	4	3	NA	
ACTIVE_CTX_STS	4	3	NA	

Table 5-4. Microengine Local CSR Latencies (Sheet 2 of 2)

Register Name	Latency			Usage Latency Comments
	Write	Read	Usage	
INDIRECT_CTX_SIG_EVENTS	4	3	NA	
ACTIVE_CTX_SIG_EVENTS	4	3	NA	
INDIRECT_CTX_WAKEUP_EVENTS	4	3	NA	
ACTIVE_CTX_WAKEUP_EVENTS	4	3	NA	
INDIRECT_CTX_FUTURE_COUNT	4	3	NA	
ACTIVE_CTX_FUTURE_COUNT	4	3	NA	
INDIRECT_LM_ADDR_0,	3	3	3	Must wait 3 cycles before issuing instruction that uses the LM_ADDR
ACTIVE_LM_ADDR_0,	3	2		
INDIRECT_LM_ADDR_1	3	3		
ACTIVE_LM_ADDR_1	3	2		
BYTE_INDEX	3	2	3	Must wait 3 cycles before issuing instruction that uses the LM_ADDR, BYTE_INDEX, or T_INDEX
ACTIVE_LM_ADDR_0_BYTE_INDEX	3	2		
INDIRECT_LM_ADDR_0_BYTE_INDEX	3	3		
INDIRECT_LM_ADDR_1_BYTE_INDEX	3	3		
ACTIVE_LM_ADDR_1_BYTE_INDEX	3	2		
T_INDEX_BYTE_INDEX	3	2		
T_INDEX	3	2		
INDIRECT_FUTURE_COUNT_SIGNAL	4	3	NA	
ACTIVE_FUTURE_COUNT_SIGNAL	4	3	NA	
NN_PUT	3	2	3	Must wait 3 cycles before issuing instruction that uses the NN_GET or NN_PUT pointer.
NN_GET	3	2		
TIMESTAMP_LOW	4	3	NA	
TIMESTAMP_HIGH	4	3	NA	
NEXT_NEIGHBOR_SIGNAL	3	2	12	If the ME writes to this CSR and the Next or Previous Neighbor is swapped out, it will wake in 12 cycles. Refer to Note 1.
PREV_NEIGHBOR_SIGNAL	3	2		
SAME_ME_SIGNAL	3	2	8	The same ME will be signaled 8 cycles after the CSR write.
CRC_REMAINDER	3	2	0	Can issue CRC ops immediate after local_csr_wr
PROFILE_COUNT	NA	NA	NA	
PSEUDO_RANDOM_NUMBER	3	2	3	When the CTX_ENABLES[PRN_MODE] bit is set to update on a read, the new PSN will be generated 3 cycles after issuing local_csr_write of PSN.
LOCAL_CSR_STATUS	NA	NA	NA	

Note 1: A write to the NEXT_NEIGHBOR_SIGNAL CSR can immediately follow a next Neighbor put or context relative write operation and the data will be guaranteed to arrive at the Next Neighbor before the signal.

5.2.1 USTORE_ADDRESS

Used to load programs into the Control Store. This register and the USTORE_DATA_LOWER and USTORE_DATA_UPPER registers are used to program the Microengine while all Contexts are in Inactive state. The Intel XScale® core writes the Control Store address to this register and follows it with reads or writes to the USTORE_DATA_UPPER and USTORE_DATA_LOWER registers.

Note: For IXP2800 Rev A only -- Before loading the Control Store write a '1' to bit 10 of Local CSR 0x154, and then write a '0' to bit 10 of Local CSR 0x154. This register is only used for manufacturing test and must be initialized prior to use.

After a write to this register, the data from the Control Store at the UADR location can be read by reading the USTORE_DATA registers.

To write the Control Store do the following (Note that ME reset signal must be deasserted before this):

1. Write the address to be written into Ustore_Address[Uadr]. Note that the ECS bit must be a '1' to write the Control Store.
2. Write the data for bits [31:0] into Ustore_Data_Lower.
3. Write the data for bits [39:32] into Ustore_Data_Upper. The write to Ustore_Data_Upper also causes the write into the Control Store, and Ustore_Addr[Uadr] to increment.
4. If writing consecutive addresses of the Ustore, repeat steps 2 and 3 for each location to be loaded. If writing to non-consecutive locations, repeat steps 1 through 3 for each location to be loaded.
5. Write the Ustore_Address[ECS] to '0' to enter normal mode.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
ECS	RESERVED															UADR														

Bits	Field	Description	RW	Reset
[31]	ECS	0: Cleared during normal execution. 1: Set when reading or writing the control store. The Microengine should be in Idle state (no contexts running). The address in Uadr field specifies the Control Store address where the data written to Ustore_Data will be written. Also set in debug mode. This bit can be used to dump data from Microengine GPRs and Read Transfer registers. The Microengine should be in an idle state (no contexts running). This forces the Microengine to continuously execute an instruction at the address specified by Uadr. Only the alu instruction is supported in this mode and the result of the execution is written to Alu_Out CSR rather than a destination register.	RW	0
[30:13]	RESERVED	Reserved	RO	0
[12:0]	UADR	Contains the address of the Control Store location to be accessed. IXP2400 and IXP2800 (Rev A): Valid values are 0 to 4095. IXP2800 (Rev B): Valid values are 0 to 8191.	RW	Undef



5.2.2 USTORE_DATA_LOWER, USTORE_DATA_UPPER

Control Store Data. These registers (along with USTORE_ADDRESS) are used to program the Microengine’s Control Store while the Microengine is in the idle state. Writing the Control Store address to USTORE_ADDRESS CSR and following it with writes to these CSRs loads an instruction (the write to USTORE_DATA_UPPER actually causes the write, and causes USTORE_ADDRESS to increment).

Reading these registers after writing USTORE_ADDRESS reads the data stored in that Control Store address

Even parity is supported on the control store which means the parity checking circuit counts the number of 1 bits across the UDATA_UPPER_1, UDATA_UPPER_2, and PAR_UP and verifies that the total number of 1 bits is an EVEN number. It also does the same across UDATA_LOWER and PAR_LOW.

Once a parity error is detected on an ME, the ME is stopped in a manner that prevents it from modifying any state, either internally or externally (so that the ME can be debugged); however the internal PC continues to increment. The recommended method for recovering from a parity error is to reset the ME.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
UDATA_UPPER_1												UDATA_LOWER																				

Bits	Field	Description	RW	Reset
[19:0]	UDATA_LOWER	Contains the lower 20 bits of the instruction of the Control Store location specified by the USTORE_ADDRESS CSR. Parity for UDATA_LOWER is provided at bits PAR_LOW.	RW	Undef
[31:20]	UDATA_UPPER_1	Contains the first 12 bits of the upper 20 bits of the instruction of the Control Store location specified by the USTORE_ADDRESS CSR. Parity for UDATA_UPPER_1 as well as UDATA_UPPER_2 is provided at bits PAR_UP.	RW	Undef

Some of the bits hold parity information, as shown in the table below. It is the responsibility of the Intel XScale® core loader routine that loads the Control Store to precompute and load the correct parity.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0					
RESERVED																						PAR_UP	PAR_LOW	UDATA_UPPER_2													

Bits	Field	Description	RW	Reset
[31:10]	RESERVED	Reserved	RO	0
[9]	PAR_UP	Parity. Contains even parity for UDATA_UPPER_1 and UDATA_UPPER_2 data of the control store location specified by the USTORE_ADDRESS CSR.	RW	Undef
[8]	PAR_LOW,	Parity. Contains even parity for UDATA_LOWER data of the control store location specified by the USTORE_ADDRESS CSR.		
[7:0]	UDATA_UPPER_2	Contains the data from bits [39:32] of the control store location specified by the USTORE_ADDRESS CSR. Contains the second 8 bits of the upper 20 bits of the instruction of the Control Store location specified by the USTORE_ADDRESS CSR. Parity for UDATA_UPPER_1 as well as UDATA_UPPER_2 is provided at bits PAR_UP.	RW	Undef

5.2.3 USTORE_ERROR_STATUS

This register captures information about parity errors during instruction reads. The contents are valid if CTX_ENABLE[CONTROL STORE PARITY ERROR] is a '1'.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED													CTX	RESERVED	UADDR															

Bits	Field	Description	RW	Reset
[31:19]	RESERVED	Reserved	RO	0
[18:16]	CTX	Context that was Executing when parity error occurred.	RO	Undef
[15:13]	RESERVED	Reserved	RO	0
[12:0]	UADDR	Contains the address that had the parity error.	RO	Undef

5.2.4 ALU_OUT

This CSR is used during debugging to access the contents of the Microengine GPRs. It can be read by the Intel XScale® core processor to view the current state of the ALU.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
ALU_OUTPUT																															

Bits	Field	Description	RW	Reset
[31:0]	ALU_OUTPUT	ALU output.	RO	Undef

5.2.5 **TIMESTAMP_HIGH, TIMESTAMP_LOW**

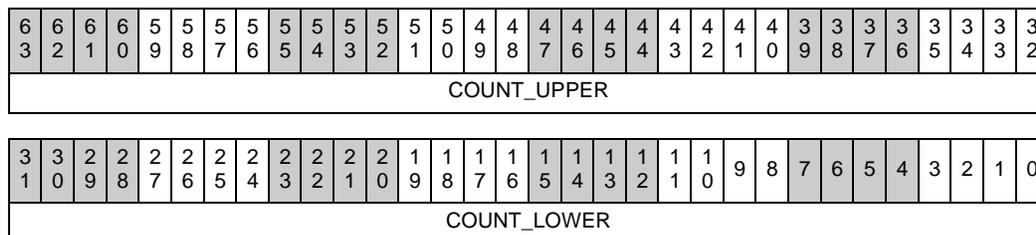
Timestamp is 64 bits. It counts up by one every sixteen cycles. When it hits the maximum value, it wraps to zero on the next count.

Note: Note that this will never happen in our lifetime.

These registers can be written to preset it to a specific value. When MISC_CONTROL[TIMESTAMP_ENABLE] (Section 5.6.4.2) is 0, Timestamp does not change so a preset value can be written and when MISC_CONTROL[TIMESTAMP_ENABLE] is set to 1, Timestamp will count normally. When MISC_CONTROL[TIMESTAMP_ENABLE] is set to 1 Timestamp cannot be written. Note that MISC_CONTROL[TIMESTAMP_ENABLE] controls all Microengines, so that all of the Timestamps can be programmed with the same value and enabled at once.

The value in Timestamp can be read. Because it is longer than 32 bits, it requires two reads, and there is a hardware feature included to insure that the two values read are consistent. [Without this feature the following sequence would be possible: 1) read lower part, 2) counter increments, and upper part changes, 3) read upper part.] When the lower part is read (at the address of TIMESTAMP_LOW), the corresponding value in the upper part is copied into a shadow register. Reads of the upper part (at the address of TIMESTAMP_HIGH) read that shadow copy instead of the actual counter.

Reading this register from the Intel XScale® core or PCI returns undefined data. ME software that reads both parts of Timestamp must read the lower part first, followed by the upper part. Note that software is not required to read both parts if only the lower part is desired; the next read of TIMESTAMP_LOW will re-load the shadow copy.



Bits	Field	Description	RW	Reset
[63:32]	COUNT_UPPER	Current count value.	RW	Undef
[31:0]	COUNT_LOWER			

5.2.6 **ACTIVE_CTX_FUTURE_COUNT**

See Section 5.2.7, “INDIRECT_CTX_FUTURE_COUNT”.

5.2.7 INDIRECT_CTX_FUTURE_COUNT

There are 8 INDIRECT_CTX_FUTURE_COUNT registers for contexts 0 through 7. The value in each is compared to the value in the low 16-bits of Timestamp, and will set the signal specified in **Future_Count_Signal** register for this Context when it is armed and there is a match of Future_Count to Timestamp. Writing to Future_Count arms the event. When the signal is set the event is disarmed (so that it will not set again when the lower half of Timestamp wraps around and matches Future_Count again).

The registers are accessed indirectly using the CSR_CTX_POINTER to select the specific context register and reading or writing the INDIRECT_CTX_FUTURE_COUNT register. The register for the context that is currently executing can be accessed by reading or writing the ACTIVE_CTX_FUTURE_COUNT Local CSR.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																FUTURE_CNT															

Bits	Field	Description	RW	Reset
[31:16]	RESERVED	Reserved.	RO	0
[15:0]	FUTURE_CNT	Value to match against low 32-bits of Timestamp.	RW	undef

5.2.8 ACTIVE_FUTURE_COUNT_SIGNAL

See Section 5.2.9, “INDIRECT_FUTURE_COUNT_SIGNAL”.

5.2.9 INDIRECT_FUTURE_COUNT_SIGNAL

There are 8 INDIRECT_FUTURE_COUNT_SIGNAL registers for contexts 0 through 7. The value in these registers indicate which signal to set when the FUTURE_COUNT is used.

The registers are accessed indirectly using the CSR_CTX_POINTER to select the specific context register and reading or writing the INDIRECT_FUTURE_COUNT_SIGNAL register. The register for the context that is currently executing can be accessed by reading or writing the ACTIVE_FUTURE_COUNT_SIGNAL Local CSR

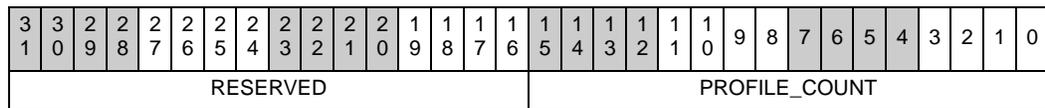
3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																										SIG_NO					

Bits	Field	Description	RW	Reset
[31:4]	RESERVED	Reserved	RO	0
[3:0]	SIG_NO	The signal number to set when FUTURE_COUNT matches TIMESTAMP.	RW	Undef

5.2.10 PROFILE_COUNT

PROFILE_COUNT is used for code profiling and tuning. It counts once per cycle, and when it reaches maximum value it wraps to 0.

A typical use of PROFILE_COUNT is to read and store the value in a register, start an external event (for example an external memory read), and then, when the external event completes, read the value in PROFILE_COUNT again. Subtracting the two values will give the elapsed number of cycles..



Bits	Field	Description	RW	Reset
[31:16]	RESERVED	Reserved	RO	0
[15:0]	PROFILE_COUNT	Count. Advances by one every cycle.	RW	Undef

5.2.11 PSEUDO_RANDOM_NUMBER

PSEUDO_RANDOM_NUMBER (PRN) uses a Linear Feedback Shift Register (LFSR) to generate a pseudo random number which can be used by Microengine software. It can be initialized by a local_csr_wr, and the number changes after each read by a local_csr_rd, according to the polynomial:

$$x^{32} + x^{31} + x^{28} + x^{27} + x^{24} + x^{17} + x^{16} + x^{14} + x^{13} + x^{12} + x^{11} + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1.$$

The period of the output sequence is $2^{32} - 1$.

Note: When a PRN is generated is based on the PRN Mode bit in the “CTX_ENABLES” CSR.

Note: The PRN will be updated if the local_csr_rd occurs in the shadow of a taken branch. This is OK as long as one just wants a random number, however if a guaranteed repeatable random sequence is desired, then do not place local_csr_rd of PRN in branch shadow.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
PSEUDO_RANDOM_NUMBER																															

Bits	Field	Description	RW	Reset
[31:0]	PSEUDO_RANDOM_NUMBER	Pseudo Random Number. This field must be initialized to a non-zero value to generate a pseudo random number.	RW	Undef

5.2.12 NEXT_NEIGHBOR_SIGNAL

The NEXT_NEIGHBOR_SIGNAL CSR allows a program to signal a Context in the Next Neighbor Microengine. The data is used to select which Context and Event Signal number is set.

This register must not be written to on two consecutive instructions. The limitation is due to the fact that the IO signals of the Microengine run at one-half the internal rate.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																							THIS_CTX	SIG_NO	CTX						

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	WO	0
[7]	THIS_CTX	Controls whether or not the Context field of this register is used in selecting the Context that is signaled in the next neighbor Microengine. 1—Signal the same numbered Context as the one that does the write to this register. This is useful if running common code on multiple Contexts that need to signal the same numbered Context. 0—Use the Context field of this register to select the Context in the next neighbor Microengine.	WO	0
[6:3]	SIG_NO	Signal to set in the next neighbor Microengine.	WO	0
[2:0]	CTX	Context to signal in the next neighbor Microengine. This field is only used if THIS_CONTEXT is not asserted	WO	0

5.2.13 PREV_NEIGHBOR_SIGNAL

The PREV_NEIGHBOR_SIGNAL CSR allows a program to signal a Context in the Previous Neighbor Microengine (the next lower numbered Microengine). The data is used to select which context and signal number is set.

This register must not be written to on two consecutive instructions. The limitation is due to the fact that the I/O signals of the Microengine run at one-half the internal rate.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						THIS_CTX	SIG_NO			CTX				

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	W	0
[7]	THIS_CTX	Controls whether or not the Context field of this register is used in selecting the Context that is signaled in the previous neighbor Microengine. 1—Signal the same numbered Context as the one that does the write to this register. This is useful if running common code on multiple Contexts that need to signal the same numbered Context. 0—Use the Context field of this register to select the Context in the previous neighbor Microengine.	W	0
[6:3]	SIG_NO	Signal to set in the previous neighbor Microengine.	W	0
[2:0]	CTX	Context to signal in the previous neighbor Microengine. This field is only used if THIS_CONTEXT is not asserted	W	0

5.2.14 SAME_ME_SIGNAL

The SAME_ME_SIGNAL CSR allows a thread to signal another Context in the same Microengine. The data is used to select which Context and signal number is set.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																							NEXT_CTX	SIG_NO				CTX			

Bits	Field	Description	RW	Reset
[31:8]	Reserved		W	0
[7]	NEXT_CTX	Controls whether or not the Context field of this register is used in selecting the Context that is signaled in the Microengine. 1—Signal the Context that is one greater (modulo the number of Active Contexts in CTX_ENABLE[IN_USE_CONTEXTS]) as the one that does the write to this register. This is useful if running common code on multiple Contexts that need to signal the next sequential Context. 0—Use the Context field of this register to select the Context in the Microengine.	W	0
[6:3]	SIG_NO	Signal to set in the Microengine.	W	0
[2:0]	CTX	Context to signal in the Microengine. This field is only used if NEXT_CONTEXT is not asserted	W	0

5.2.15 ACTIVE_CTX_STS

This register maintains the context number of the Context currently executing.

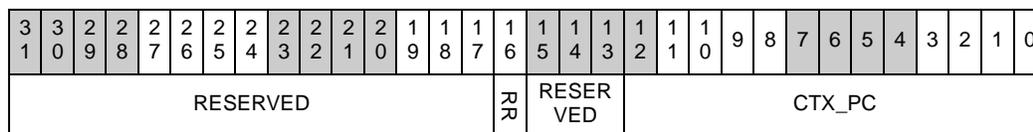
Each Microengine supports eight contexts (0 through 7).

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
ABO	RESERVED										ACTXPC										ME_NO				ACNO						

Bits	Field	Description	RW	Reset
[31]	AB0	If set, the Microengine has a Context in the Executing state. If clear, no Context is in Executing State.	RO	0
[30:21]	RESERVED	Reserved	RO	0
[20:8]	ACTXPC	PC of Executing Context. Only valid if AB0 is a 1. IXP2400 and IXP2800 (Rev A): Valid values are 0 to 4095. IXP2800 (Rev B): Valid values are 0 to 8191. This field provides a snapshot value of the PC. This value is used for tracking/code profiling purposes. When issued as a local_csr_read from the ME, the PC value may not be the exact PC value of the local_csr_rd instruction.	RO	0
[7:3]	ME_N0	The number of this Microengine. [7] = ME cluster [6:3] = ME number	RO	Note
[2:0]	ACNO	The number of the Executing context. Only valid if AB0 bit is a 1.	RW	0
Note -- the reset value is the number of the Microengine.				

5.2.16 INDIRECT_CTX_STS

There are 8 INDIRECT_CTX_STS registers, each contain the status of contexts 0 through 7. These registers are accessed indirectly using the CSR_CTX_POINTER to select the specific context register and reading or writing this register. Note that the active context number can be read via ACTIVE_CTX_STS Local CSR.



Bits	Field	Description	RW	Reset
[31:17]	RESERVED	Reserved	RO	0
[16]	RR	Ready to Run. Indicates that the Context is in Ready state. (This bit will be 0 if the Context is in any of the other three states).	RO	Undef
[15:13]	RESERVED	Reserved	RO	0
[12:0]	CTX_PC	The program counter at which the Context begins executing when it is put into the Executing state. IXP2400 and IXP2800 (Rev A): Valid values are 0 to 4095. IXP2800 (Rev B): Valid values are 0 to 8191.	RW	Undef

5.2.17 CTX_ARB_CNTL

This register is used by the context arbiter and is also used for debugging.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																								PCTX		RESERVED	NCTX			

Bits	Field	Description	RW	Reset
[31:7]	RESERVED	Reserved	RO	0
[6:4]	PCTX	Previous Context. This field contains the number of the last context that was running.	RO	Undef
[3]	RESERVED	Reserved	RO	0
[2:0]	NCTX	Next Context. This field contains the number of the next context that will be run.	RW	0

5.2.18 CTX_ENABLES

This register is used by the context arbiter and is also used for debugging.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0		
IN_USE_CTX	PRN_Mode	CTL_STR_PAR_ER	CTL_STR_PAR_EN	BREAKPOINT	RESERVED							NN_MODE	NN_RING_EMPTY	LM_ADDR_1_GLOB	LM_ADDR_0_GLOB	CCTX_ENABLES										RESERVED						

Bits	Field	Description	RW	Reset
[31]	IN_USE_CTX	Indicates the number of in-use contexts, which determines the GPR and Transfer Register allocation. Note that although this information could be inferred from bits C0 to C7, this field allows for contexts to be temporarily disabled due to error or debug conditions. 0— 8 Context mode (Contexts 0, 1, 2, 3, 4, 5, 6, 7) 1— 4 Context mode (Contexts 0, 2, 4, 6) It is illegal to enable Contexts that are not currently in-use according to this field. In 4 context mode the SRAM, scratch, cap, PCI, and MSF instructions can only read data into the S-transfer registers and not either the S or D transfer registers.	RW	Undef
[30]	PRN_MODE	Controls when the Pseudo_Random_Number is generated. 0—Pseudo_Random_Number is updated only when it is loaded or read using the local_CSR instruction (not when the Intel XScale® core or PCI reads the local CSR). 1—Pseudo_Random_Number is updated every cycle	RW	Undef
[29]	CTL_STR_PAR_ER	Indicates that a parity error was detected in the Control Store when an instruction was read. This bit will never be set if CONTROL STORE PARITY ENABLE bit is 0. When this bit is set the Microengine's attn output is asserted. The handling of such a parity error should involve resetting the corresponding micro-engine, instead of clearing this bit. This is because the micro-engine may have fetched invalid code or may have incorrectly updated its program counter.	RW 1C	0
[28]	CTL_STR_PAR_EN	Enables parity checking on Control Store. 0—Parity checking disabled, no error possible. 1—Parity checking enabled.	RW	0
[27]	BREAKPOINT	ctx_arb[bpt] instruction was executed. When this bit is set: 1. The Microengine's attn output is asserted. 2. All CTX_ENABLES bits in this register are cleared.	RW 1C	0
[26:21]	RESERVED	Reserved	RO	0

Bits	Field	Description	RW	Reset
[20]	NN_MODE	<p>Controls how the Next Neighbor Registers in this ME are written.</p> <p>0—From Previous Neighbor Microengine 1—From this Microengine</p> <p>The Next Neighbor Registers can only be written by one source. If ME(n+1) has this bit set and ME(n) has this bit clear and ME (n) executes an instruction that uses the next neighbor registers as a destination, the instruction is executed but the result is not written anywhere. This bit setting for Next Neighbor MEs can be summarized as follows:</p> <p>ME(n) = ME(n+1) = 0: Each ME writes to the next neighbor's NN registers</p> <p>ME(n) = 0, ME(n+1) = 1: ME0 NN writes are not written anywhere ME1 writes to its own NN registers</p> <p>ME(n) = 1, ME(n+1) = 0: ME0 writes to its own NN registers ME1 writes to next neighbors NN registers</p> <p>ME(n) = ME(n+1) = 1: Each ME writes there own NN registers</p> <p>When this bit is set to 1 the ME can use its NN registers in context relative or ring mode and the ring full signals can be used to get the status of its own NN Ring.</p> <p>When an ME writes to its own Neighbor register it must wait 5 cycles (or instructions) before it executes the instruction that reads the same register in order to get the newly written value.</p>	RW	Undef
[19:18]	NN_RING_EMPTY	<p>Controls threshold when NN_Empty asserts. The number of entries valid is determined by comparing NN_PUT and NN_GET Local CSRs. The use of indicating empty when there is really something on the Ring is if the cooperating processes transfer data in a block, and the consumer does not want to get a partial block.</p> <p>00—0 entries valid 01—1 or less entries valid 10—2 or less entries valid 11—3 or less entries valid</p>	RW	Undef
[17]	LM_ADDR_1_GLOB	<p>Controls usage of LM_ADDR_1.</p> <p>0—LM_ADDR_1 is Context Relative. Each Context uses a separate copy of LM_ADDR_1. 1—LM_ADDR_1 is Global. Only the working copy of LM_ADDR_1 is used, independent of Active Context.</p>	RW	Undef
[16]	LM_ADDR_0_GLOB	<p>Controls usage of LM_ADDR_0.</p> <p>0—LM_ADDR_0 is Context Relative. Each Context uses a separate copy of LM_ADDR_0. 1—LM_ADDR_0 is Global. Only the working copy of LM_ADDR_0 is used, independent of Active Context.</p>	RW	Undef
[15:8]	CTX_ENABLES	<p>Context Enables for Context 7 through Context 0.</p> <p>0—Context is in Inactive state. 1—Context may be in Ready, Executing, or Sleep states.</p>	RW	0
[7:0]	RESERVED	Reserved	RO	0

5.2.19 CC_ENABLE

The condition codes are always enabled for normal use and are disabled only during ME debugging where the GPRs are read by an external source (PCI, Intel XScale® core, other ME) and condition codes need to be preserved.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																	CCCE	RESERVED										PMU_ENABLES	PMU_CTX_MON			

Bits	Field	Description	RW	Reset
[31:14]	RESERVED	Reserved	RO	0
[13]	CCCE	Current Condition Code Enable. Set to 1 to update the condition codes. When 0, condition codes will not be updated.	RW	Undef
[12:4]	RESERVED	Reserved	RO	0
[3]	PMU_ENABLE	IXP2400 and IXP2800 Rev A -- Reserved IXP2800 Rev B -- In order for the PMU (Performance Monitor Unit) to be operational in the Microengine, the programmer must write a 1 to the PMU_Enable bit.	RW	0
[2:0]	PMU_CTX_MON	This field holds the number of the Context to be monitored. The event count will only reflect events that occurred when this Context is Executing.	RW	Undef

5.2.20 CSR_CTX_POINTER

This register is used when reading or writing the Local CSRs that are unique per Context. The PCI, Intel XScale® core or ME writes a context number into this register and then reads or writes any Local CSR with a name that starts with “INDIRECT_” to access the version of the register that is specific to the context.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																												CTX				

Bits	Field	Description	RW	Reset
[31:3]	RESERVED	Reserved	RO	0
[2:0]	CTX	Selects which contexts Local CSR is accessed by local_csr_read, local_csr_write, and by the Intel XScale® core.	RW	Undef

5.2.21 ACTIVE_CTX_SIG_EVENTS

See Section 5.2.22, “INDIRECT_CTX_SIG_EVENTS”

5.2.22 INDIRECT_CTX_SIG_EVENTS

There are 8 INDIRECT_CTX_SIG_EVENTS registers, each contain status information that indicates which Event Signals have occurred for contexts 0 through 7. The registers are accessed indirectly using the CSR_CTX_POINTER to select the specific context register and reading or writing the INDIRECT_CTX_SIG_EVENTS register. The register for the context that is currently executing can be accessed by reading or writing the ACTIVE_CTX_STS Local CSR.

This register is used in conjunction with the CTX_WAKEUP_EVENTS register to move the Context from Sleep state to Ready state.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED															SIGNAL_15...SIGNAL_1											VOL		

Bits	Field	Description	RW	Reset
[31:16]	RESERVED	Reserved	RO	0
[15:1]	SIGNAL_15... SIGNAL_1	Each bit is set as described in the Event Signals section. Each is cleared by Microengine hardware when either: <ul style="list-style-type: none"> the signal is used to transition to Ready state if the CTX_WAKEUP_EVENT OR bit is clear. a br_!signal on this signal is not taken. a br_=signal on this signal is taken. 	RW	Undef
[0]	VOL	Corresponds to Event for Voluntary arb wakeup event.	RO	1

5.2.23 ACTIVE_CTX_WAKEUP_EVENTS

See Section 5.2.24, “INDIRECT_CTX_WAKEUP_EVENTS”

5.2.24 INDIRECT_CTX_WAKEUP_EVENTS

There are 8 INDIRECT_CTX_WAKEUP_EVENTS registers, each contain status information that indicate which Event Signals are required to put the each of the Contexts into Ready state. The registers are accessed indirectly using the CSR_CTX_POINTER to select the specific context



register and reading or writing the INDIRECT_CTX_WAKEUP_EVENTS register. The register for the context that is currently executing can be accessed by reading or writing the ACTIVE_CTX_WAKEUP_EVENTS Local CSR.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED													ANY_WK_EVENTS	WAKEUP_EVENT_15...WAKEUP_EVENT_1											VOLUNTARY	

Bits	Field	Description	RW	Reset
[31:17]	RESERVED	Reserved	RO	0
[16]	ANY_WK_EVENTS	ANY Wakeup Events. Set by the ANY token on a <code>ctx_arb</code> instruction, 0—All mode (AND) 1—ANY mode (OR)	RW	Undef
[15:1]	WAKEUP_EVENT_15... WAKEUP_EVENT_1	Each wakeup event bit is set by either a <code>ctx_swap_#</code> token on an instruction, or by the Event Signal Mask of the <code>ctx_arb</code> instruction. All wakeup event bits are cleared by Microengine hardware whenever the Context is put into Execute state.	RW	Undef
[0]	VOLUNTARY	Set by <code>ctx_arb[voluntary]</code> . Cleared when the Context is put into Execute state.	RW	Undef

5.2.25 ACTIVE_LM_ADDR_0

See “INDIRECT_LM_ADDR_1”.

5.2.26 ACTIVE_LM_ADDR_1

See “INDIRECT_LM_ADDR_1”.

5.2.27 INDIRECT_LM_ADDR_0

See “INDIRECT_LM_ADDR_1”.

5.2.28 INDIRECT_LM_ADDR_1

These registers hold the addresses which are used to read and write Local Memory (LM). Each Context has its own LM_ADDR_0 and LM_ADDR_1. There is also a working copy of each. When a Context is put into Executing state, the value from its pair of LM_ADDRS are copied into the working pair.

Reads or writes of ACTIVE_LM_ADDR_# select the working pair.

The context relative registers are accessed indirectly using the CSR_CTX_POINTER to select the specific context register and reading or writing the INDIRECT_LM_ADDR_0 and INDIRECT_LM_ADDR_1 registers. The working registers can be accessed by reading or writing the ACTIVE_LM_ADDR_0 and ACTIVE_LM_ADDR_1 CSR. When the Context goes to Sleep state, the value in the working pair is moved to the Context Specific pair.

When a LM_ADDR is being used globally (as set in CTX_ENABLE[LM_ADDR_GLOBAL]), the working pair is used and never overwritten during Context swaps.

The working LM_ADDR can also be loaded with the result of a lookup_cam instruction.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																		LM_ADDR										RESERVED				

Bits	Field	Description	RW	Reset
[31:12]	Reserved		RO	0
[11:2]	LM_ADDR	Selects the specific 32-bit word in Local Memory. This field can be incremented or decremented by 1, or left unchanged after the access, as specified in the instruction.	RW	Undef
[1:0]	RESERVED	Reserved	RO	0

5.2.29 BYTE_INDEX

This register is used to control the byte shift amount during Byte_Align instructions. This register can be written alone, or can be written along with T_INDEX or LM_ADDR (see Section 5.2.31, “T_INDEX_BYTE_INDEX”) and INDIRECT_LM_ADDR_#_BYTE_INDEX (see Section 5.2.32, “INDIRECT_LM_ADDR_0_BYTE_INDEX”).

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																												BYTE_NO				

Bits	Field	Description	RW	Reset
[31:2]	Reserved		RO	0
[1:0]	BYTE_NO	Specifies a byte number for use with the byte_align instruction.	RW	Undef

5.2.30 T_INDEX

This register is used when S_TRANSFER or D_TRANSFER registers are accessed via indexed mode, which is specified in the source and destination fields of the instruction. This register can be incremented, decremented, or left unchanged after the access. This register can be written alone, or can be written along with BYTE_INDEX (see Section 5.2.31, “T_INDEX_BYTE_INDEX”).

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																							XFER_INDEX					RESERVED				

Bits	Field	Description	RW	Reset
[31:9]	RESERVED	Reserved	RO	0
[8:2]	XFER_INDEX	Transfer Register Index. Specifies one of 128 registers. The choice of S_TRANSFER_IN, S_TRANSFER_OUT, D_TRANSFER_IN, D_TRANSFER_OUT is made by the other bits of the register specifier, and the use (either source or destination). This field can be incremented or decremented by 1, or left unchanged after the access, as specified in the instruction.	RW	Undef
[1:0]	RESERVED	Reserved	RO	0

5.2.31 T_INDEX_BYTE_INDEX

This is an alias of the Section 5.2.30, “T_INDEX” and Section 5.2.29, “BYTE_INDEX” register. Reading and writing this register reads and writes both the T_INDEX and BYTE_INDEX registers.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																							XFER_INDEX					BYTE_NO				

Bits	Field	Description	RW	Reset
[31:9]	RESERVED	Reserved	RO	0
[8:2]	XFER_INDEX	Section 5.2.30, “T_INDEX”	RW	Undef
[1:0]	BYTE_NO	Section 5.2.29, “BYTE_INDEX”	RW	Undef

5.2.32 INDIRECT_LM_ADDR_0_BYTE_INDEX

This is an alias of the [Section 5.2.27, “INDIRECT_LM_ADDR_0”](#) and [Section 5.2.29, “BYTE_INDEX”](#) register. Reading and writing this register reads and writes both the LM_ADDR_0 and BYTE_INDEX registers.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																		LM_ADDR								BYTE_NO					

Bits	Field	Description	RW	Reset
[31:12]	RESERVED	Reserved	RO	0
[11:2]	LM_ADDR	Section 5.2.27, “INDIRECT_LM_ADDR_0”	RW	Undef
[1:0]	BYTE_NO	Section 5.2.29, “BYTE_INDEX”	RW	Undef

5.2.33 INDIRECT_LM_ADDR_1_BYTE_INDEX

See [Section 5.2.32, “INDIRECT_LM_ADDR_0_BYTE_INDEX”](#).

5.2.34 ACTIVE_LM_ADDR_0_BYTE_INDEX

See [Section 5.2.32, “INDIRECT_LM_ADDR_0_BYTE_INDEX”](#).

5.2.35 ACTIVE_LM_ADDR_1_BYTE_INDEX

See [Section 5.2.32, “INDIRECT_LM_ADDR_0_BYTE_INDEX”](#).

5.2.36 NN_PUT

This register contains the “put” pointer used when the Next Neighbor registers are used as a Ring. When “previous” ME executes an instruction that specifies the destination as *n\$index, the Next Neighbor register in “this” ME is selected is by the value in this register, and the value is then incremented by one (a value of 127 wraps back to 0). The value in this register is compared to the value in NN_GET register to determine when to assert NN_FULL and NN_EMPTY status signals. There is a behavior of the Ring is shown in [Table 5-5](#) and [Table 5-6](#).

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						NN_REG_INDEX									



Bits	Field	Description	RW	Reset
[31:7]	RESERVED	Reserved	RO	0
[6:0]	NN_REG_INDEX	Specifies one of 128 NN registers to write.	RW	Undef

Table 5-5. NN_PUT Ring Behavior

Description	# Entries
MAX allowable entries in Neighbor Ring	124
NN_FULL asserts when number of entries	96
NN_FULL asserts when number of available entries (note 1)	28
Cycle latency between a Neighbor put and the update of NN_FULL status	20
Note 1: Equals 124 - 96	

Table 5-6. NN_PUT Ring Latency

Number of cycles between a Neighbor put and BR_INP_STATE[NN_FULL,label#]	Number of longwords that can be safely put into a Neighbor without overflowing the NN_RING
0	8
1	9
...	...
19	27
20	28
21	28
22	28

To understand this table consider the example where there are currently 95 LWs in the NN ring and software reads the full state as "not full". Since it takes 20 cycles to get the NN put data into the ring, software must assume (conservatively) that there are 20 put operations currently in progress. If the software waits 0 cycles between testing the full state and the put operations software can safely perform 8 put operations (28 - 20). On the other hand, if the software waits 20 cycles between testing the full state and the put operations, any put operation in progress will have completed, and software can safely perform 28 put operations.

5.2.37 NN_GET

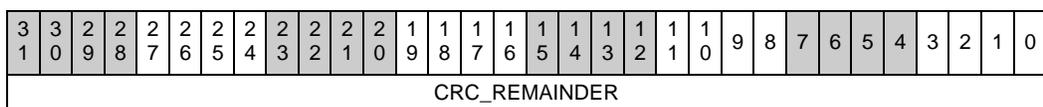
This register contains the “get” pointer used when the Next Neighbor registers are used as a Ring. This register is used to specify the Next Neighbor register when a source operand is *n\$index, and the value is then incremented (a value of 127 wraps back to 0). The value in this register is compared to the value in NN_PUT register to determine when to assert NN_FULL and NN_EMPTY status signals.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																							NN_REG_INDEX										

Bits	Field	Description	RW	Reset
[31:7]	Reserved		RO	0
[6:0]	NN_REG_INDEX	Specifies one of 128 NN registers to read.	RW	Undef

5.2.38 CRC_REMAINDER

CRC_REMAINDER provides one operand to the CRC Unit, and gets written with the result of the CRC operation after a crc instruction.



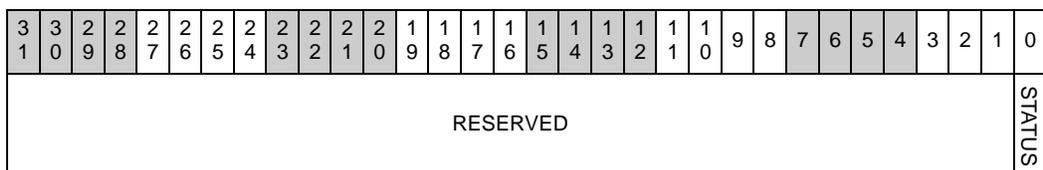
Bits	Field	Description	RW	Reset
[31:0]	CRC_REMAINDER	Input operand and result of crc instruction.	RW	Undef

5.2.39 LOCAL_CSR_STATUS

This register is used by the Intel XScale® processor or the PCI to determine if they successfully accessed an MEs Local CSR.

The Local CSRs are single ported but can be accessed by the Local ME or a remote source. The remote sources include: a Remote ME, Intel XScale® core, or PCI which use CAP to get access. If only one source accesses a given Local CSR, the access will complete as desired. If more than one source attempts to access a Local CSR on the same cycle, the Local ME will win arbitration, and a sticky status flag will be set to indicate a collision. The collision will also occur if the ME and the external source are accessing two different registers. If the external access was a read, the data returned to the Remote ME, Intel XScale® core or PCI will be unpredictable; if the external access was a write, that write will not occur. The LOCAL_CSR_STATUS register can be read by the Remote ME, Intel XScale® core or PCI to determine if the access completed without a collision. Since CAP is used to access the Local CSRs, only one remote source (CAP) will ever request access on a given cycle.

If there was a collision, the access should be repeated and status check again. Since the flag is sticky, several accesses can be done prior to testing the flag. A read of LOCAL_CSR_STATUS clears the flag. Note that LOCAL_CSR_STATUS Register is not part of the arbitrated Local CSR port, rather it is only used for the Remote ME, Intel XScale® core or PCI and can always be read predictably.



Bits	Field	Description	RW	Reset
[31:1]	Reserved		RO	0
[0]	STATUS	<p>Indicates the status of Local CSR access in the Microengine by the Intel XScale® core</p> <p>0—No accesses to Local CSRs were lost since last reading this register</p> <p>1—One or more accesses to Local CSRs were lost due to collisions with Local CSR accesses by Microengine code.</p> <p>Note that this bit is cleared by a read of this CSR.</p>	RC	0

5.3 RDR DRAM Controller - IXP2800

Table 5-7 shows the offset addresses of the DRAM Controller registers. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can be accessed Intel XScale® core and PCI (not the ME).

The IXP2800 supports three RDR DRAM Channels, each with their own register set.

Table 5-7. RDR DRAM Register Summary

Register	Offset (Hex)	Comment	Section
RDRAM_CONTROL	0x00		Section 5.3.1
RDRAM_ERROR_STATUS_1	0x08		Section 5.3.2
RDRAM_ERROR_STATAS_2	0x10		Section 5.3.3
RDRAM_ECC_TEST	0x18		Section 5.3.4
RDRAM_SERIAL_COMMAND	0x20		Section 5.3.5
RDRAM_SERIAL_DATA	0x28		Section 5.3.6
RDRAM_CONFIG_1	0x30		Section 5.3.7
RDRAM_CONFIG_2	0x38		Section 5.3.8
RDRAM_CONFIG_3	0x40		Section 5.3.9
RDRAM_RAC_INIT	0x48		Section 5.3.10
RDRAM_MISC_RAC_CONTROL	0x50		Section 5.3.11
RDRAM_RAC_CONFIG	0x58		Section 5.3.12
RDRAM_1066_CONFIG_GROUP	0x60		Section 5.3.13
RDRAM_SERIAL_CONFIG	0x68		Section 5.3.14
K0 through K11	0x100-0x158		Section 5.3.15

All RDRAMs connected on a channel must have the same timing, size, and bank properties. RDRAMs on different channels may have different properties, but the number of addresses on all populated channels must be the same.

5.3.1 RDRAM_CONTROL (# = 0,1,2)

The # symbol in the register name signifies the DRAM Channel. This register controls parameters for the channel. The proper values to use in this register will be determined by the RDRAMs used in the system.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
UNUSED_SPARE													BANK_REMAP	RD_BP_DIS	ECC	UNUSED_SPARE	NO_CHAN	SIZE	ADDRESS_REMAP	UNUSED_SPARE									

Bits	Field	Description	RW	Reset
[31:19]	UNUSED_SPARE	Reserved	RO	0
[18:17]	BANK_REMAP	IXP2800 Rev A -- Select remapping of the address according to Table 5-8 . IXP2800 Rev B -- if ADDRESS_REMAP is configured for Optimize RDRAMs Select remapping of the address according to Table 5-8 if ADDRESS_REMAP is configured for Optimize Banks Select remapping of the address according to Table 5-9	RW	0
[16]	RD_BP_DIS	Read Back pressure Disable. Back pressure should always be enabled (bit set to 0) 1 back pressure disabled for read commands 0 back pressure enabled default value	RW	
[15:14]	ECC	Controls the protection mode used. 00—None 01—Parity 10—ECC 11—Reserved	RW	1
[13]	UNUSED_SPARE	Reserved	RW	0
[12:11]	NO_CHAN	The NO_CHAN bits in each of the three RAMBUS controllers are configured as a set to select the number of RAMBUS controllers in use. All the channels are enabled initially and software must disable the desired channels. For proper operation, the channels must be disabled by writing to this field in the order: CH2, CH1, CH0. These bits should be set as follows: 3 Channels Enabled (CH0, CH1, CH2): CH2 = 0b10, CH1 = 0b10, CH0 = 0b10 2 Channels Enabled (CH0, CH1): CH2 = 0b00, CH1 = 0b01, CH0 = 0b01 1 Channel Enabled (CH0): CH2 = 0b00, CH1 = 0b00, CH0 = 0b00	RW	0x2

Bits	Field	Description	RW	Reset
[10:8]	SIZE	Rev A -- Indicates how much memory is on the channel. This controls the interleave remapping. 000—8 MB 001—16 MB 010—32 MB 011—64 MB 100—128 MB 101—256 MB 110—512 MB 111—1 GB	RW	1
[7:6]	ADDRESS_REMAP	IXP2800 Rev A -- Reserved IXP2800 Rev B.-- 00- Optimize RDRAMs -- This setting can be used when the number of memory devices on the channel is a power of two. The address is remapped according to Table 5-8 to optimize RDRAM chip selection. 01 - Optimize Banks -- This setting should be used when the number of memory devices on the channel is not a power of two (for example, it there are 3 RDRAM chips). The address is remapped according to Table 5-9 to optimize RDRAM bank selection. 10 - No Remap -- This setting disables address remapping. 11 - Reserved	RW	0
[5:0]	UNUSED_SPARE	Reserved	RW	0

Table 5-8. Address Bank Remapping (Optimize RDRAMs)

Memory Size on Channel (MB) ³	Remapped Address for IXP2800 Rev A and IXP2800 Rev B when RDRAM_CONTROL[ADDRESS_REMAP] is configured for Optimize RDRAMs Based on RDRAM_Control[Bank_Remap]			
	00	01	10	11
8	7:14, 22:15	9:14, 7:8, 22:15	11:14, 7:10, 22:15	13:14, 7:12, 22:15
16	7:14, 23:15	9:14, 7:8, 23:15	11:14, 7:10, 23:15	13:14, 7:12, 23:15
32	7:14, 24:15	9:14, 7:8, 24:15	11:14, 7:10, 24:15	13:14, 7:12, 24:15
64	7:14, 25:15	9:14, 7:8, 25:15	11:14, 7:10, 25:15	13:14, 7:12, 25:15
128	7:14, 26:15	9:14, 7:8, 26:15	11:14, 7:10, 26:15	13:14, 7:12, 26:15
256	7:14, 27:15	9:14, 7:8, 27:15	11:14, 7:10, 27:15	13:14, 7:12, 27:15
512	7:14, 28:15	9:14, 7:8, 28:15	11:14, 7:10, 28:15	13:14, 7:12, 28:15

Table 5-8. Address Bank Remapping (Optimize RDRAMs)

Memory Size on Channel (MB) ³	Remapped Address for IXP2800 Rev A and IXP2800 Rev B when RDRAM_CONTROL[ADDRESS_REMAP] is configured for Optimize RDRAMs Based on RDRAM_Control[Bank_Remap]			
	00	01	10	11
1024	7:14, 29:15	9:14, 7:8, 29:15	11:14, 7:10, 29:15	13:14, 7:12, 29:15
Bits used to select Bank Command FIFO	7:8	9:10	11:12	13:14

NOTES:

1. Table shows device/bank sorting of the channel remapped block address, which is in address 31:7. LSBs of the address are always 6:0 (byte within the block), which are not remapped
2. Unused MSBs of address have value of 0.
3. Size is programmed in RDRAM_CONTROL[SIZE].

Table 5-9. Address Bank Remapping (Optimize Banks)

Remapped Address when RDRAM_CONTROL[ADDRESS_REMAP] is configured for Optimize banks Based on RDRAM_Control[Bank_Remap]			
00	01	10	11
29:24, 7:14, 23:15	29:24, 9:14, 7:8, 23:15	29:24, 11:14, 7:10, 23:15	29:24, 13:14, 7:12, 23:15

NOTES:

1. Table shows device/bank sorting of the channel remapped block address, which is in address 31:7. LSBs of the address are always 6:0 (byte within the block), which are not remapped
2. Bits [31:30] have a value of 0.
3. Accessing an address beyond the size of the physical memory on the channel will produce unpredictable results.

5.3.2 RDRAM_ERROR_STATUS_1 (# = 0,1,2)

The # symbol in the register name signifies the DRAM Channel. This register records the type and address of a location with a parity or ECC error. It always loads the address of the most recent location accessed. If either a Correctable or Uncorrectable ECC error is detected, the value in the register is locked (it is locked when either of the ECC error bits is set), with one exception—an Uncorrectable ECC error that occurs while the register is locked will overwrite the address of the Correctable ECC error. The address of a subsequent error will be lost until software clears the error bits.

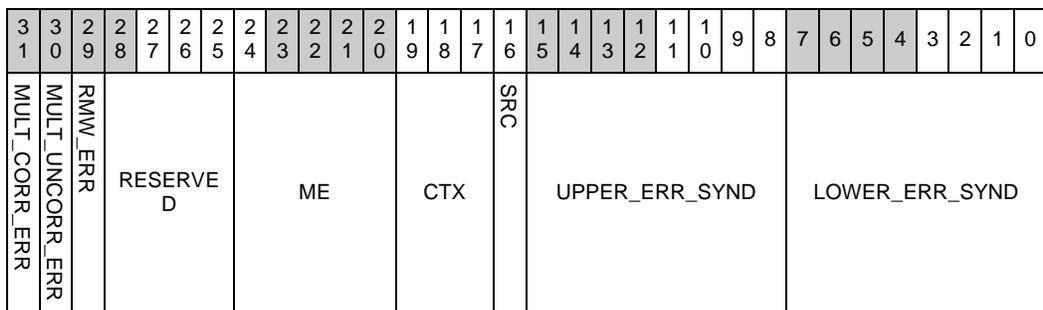
3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED	ERR_ADDR																									RESERVED	UNCOR_ERR	COR_ERR			

Bits	Field	Description	RW	Reset
[31]	RESERVED	Reserved. Read only as 0.	RO	0
[30:4]	ERR_ADDR	Address of the location with error. Only valid if either Corr_Error or Uncorr_Error is asserted.	RO	0
[3:2]	RESERVED	Reserved	RO	0
[1]	UNCOR_ERR	A parity or uncorrectable ECC error was detected. Will interrupt the Intel XScale® core if enabled. (See the Note below.)	W1C	0
[0]	COR_ERR	A correctable ECC error was detected. Will interrupt the Intel XScale® core if enabled. (See the Note below.)	W1C	0

NOTE: It is possible for both the UNCOR_ERR and COR_ERR to be set. This may occur if more than 2 bits are in error. In this case the user should assume that an uncorrectable error has occurred.

5.3.3 RDRAM_ERROR_STATUS_2 (# = 0,1,2)

The # symbol in the register name signifies the DRAM Channel. This register records the byte(s) with parity errors, or syndrome of an ECC error, and the originator of the read. It always loads the information of the most recent location accessed. If either a Correctable or Uncorrectable error is detected, the value in the register is locked (it is locked when either of the error status bits is set), with one exception—an Uncorrectable ECC error that occurs while the register is locked will overwrite the information of the Correctable ECC error. This implies that once an Uncorrectable Error occurs the information of a subsequent error will be lost.



Bits	Field	Description	RW	Reset
[31]	MULT_CORR_ERR	MULTIPLE CORR ERROR. Indicates that multiple correctable errors have occurred. This bit is reset when the Corr_Error bit in Error_Status_1 is cleared.	RO	0
[30]	MULT_UNCORR_ERR	MULTIPLE UNCORR ERROR. Indicates that multiple uncorrectable errors have occurred. This bit is reset when the Uncorr_Error bit in Error_Status_1 is cleared.	RO	
[29]	RMW_ERROR	Indicates that the Error information captured was from an RMW operation	RO	
[28:25]	RESERVED	This field is for internal use only. When read, the value of this field will vary.	RO	0 Note 1

Bits	Field	Description	RW	Reset
[24:20]	ME	Indicates which ME was the originator of the transaction that had an error. Only valid if Source is a 1. Note -- if ME, and indirect_ref is used to direct the read data to a different ME, this field is that ME. If data is sent to TBUF, this field is the ME that originated the command.	RO	0
[19:17]	CTX	Indicates the originator of the transaction (based on the Source bit) that had an error. If Source is 1, this field is the ME Context number, if Source is 0 this field is Intel XScale® core or PCI as follows. 000—Intel XScale® core 001—PCI Others—Reserved Note -- if ME, and indirect_ref is used to direct the read data to a different Context, this field is that Context. If data is sent to TBUF, this field is the ME that originated the command.	RO	0
[16]	SRC	Source. Indicates, in conjunction with the Thread field the originator of the transaction (either a read or partial write that caused a read-modify-write) that had an error. 0—Intel XScale® core or PCI 1—Microengine	RO	0
[15:8]	UPPER_ERR_SYND	UPPER ERROR SYNDROME. Byte(s) with parity errors, or syndrome of an ECC error from data bits[127:64].	RO	0
[7:0]	LOWER_ERR_SYND	Byte(s) with parity errors, or syndrome of an ECC error from data bits[63:0].	RO	0
1. The reset value of bits 27:25 on Channel 1 is 0x7				

5.3.4 RDRAM_ECC_TEST (# = 0,1,2)

The # symbol in the register name signifies the DRAM Channel. This register can be used to force incorrect byte parity or ECC code into the memory for test and diagnostic purposes. Each bit, when set, will invert the corresponding generated check bit during writes; there is no effect on reads.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																				INV_CHK_BIT											

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	INV_CHK_BIT	<p>INVERT CHECK BIT. Each bit controls one of the ecc check bits during writes.</p> <p>0 = Normal operation. 1 = Invert ecc during writes.</p> <p>The correspondence between bits in this register and ecc check bits is:</p> <p>Bit 0—ECC[0] Bit 4—ECC[4] Bit 1—ECC[1] Bit 5—ECC[5] Bit 2—ECC[2] Bit 6—ECC[6] Bit 3—ECC[3] Bit 7—ECC[7]</p> <p>The behavior of the RDRAM controller will be unpredictable if more than 2 bits are set in this register at a time for uncorrectable error or more than 1bit is set at a time for correctable errors.</p>	RW	0

5.3.5 RDRAM_SERIAL_COMMAND (# = 0,1,2)

The # symbol in the register name signifies the DRAM Channel. This register is used to initiate serial commands to the RDRAMs. The command is started when the register is written.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
BUSY	INTEL_PRIVATE										SER_ADDR										SER_BCAST	SER_DEV					SER_OP					

Bits	Field	Description	RW	Reset
[31]	BUSY	When a 1 this bit indicates that a serial command is in progress this register and RDRAMN_SERIAL_DATA must not be written in that case. When the command completes this bit reads as a 0.	RO	0
[30:23]	INTEL_PRIVATE	Intel Private and should always be written with 0.	RO	0
[22:11]	SER_ADDR	Serial address. This field is used in the SA field to select which control register of the selected RDRAM device is read or written.	RW	0
[10]	SER_BCAST	Serial broadcast. This bit is used in the SBC field; when set, RDRAM devices ignore SDEV for RDRAM device selection.	RW	0
[9:4]	SER_DEV	Serial device. This field is used in the SDEV field to select the RDRAM device to which the transaction is directed.	RW	0
[3:0]	SER_OP	Serial Op. This field is used as the Serial Op on D_CMD.	RW	0

5.3.6 RDRAM_SERIAL_DATA (# = 0,1,2)

The # symbol in the register name signifies the DRAM Channel. This register holds the data for the RDRAM serial port. For a serial port write, this register must be written prior to writing RDRAM_SERIAL_COMMAND. For a serial port read, this register holds the data from the RDRAM. The read data will be valid after a read when RDRAM_SERIAL_COMMAND[BUSY] goes to 0.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
SRD																SER_DATA															

Bits	Field	Description	RW	Reset
[31:16]	SRD	Serial Read Data. The 16 bits of data read from the selected control register of the selected RDRAM device.	RO	0
[15:0]	SWD	Serial Write Data. The 16 bits of data written to the selected control register of the selected RDRAM device.	WO	0

5.3.7 RDRAM_CONFIG_1 (# = 0,1,2)

The # symbol in the register name signifies the DRAM Channel. This register holds configuration parameters for timing and other controls.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
B_TO_B_RD	CC_TEST	UNUSED_SPARE	TRPREFSYN	TRASREFSYN	STEP2_RAC_CC_EN	TOFFP	TRP	RESERVED	TRASSYN	TCAC	TRCD																				

Bits	Field	Description	RW	Reset
[31]	B_TO_B_RD	Back-to-back RD 1: Back-to-back RD packets may be issued to different RDRAMs 0: There must be at least one SynCk between RD packets to different RDRAMs.	RW	0

Bits	Field	Description	RW	Reset
[30:28]	CC_TEST	<p>Selects modes for shortening time to test the 24-bit CC interval counter (CCINTCNT).</p> <p>CCINTCNT is split into three 8-bit subcounters: CCINTCNTA, CCINTCNTB and CCINTCNTC:</p> <p>Clocked Compared Notes</p> <p>000 23:0 23:16 Normal mode</p> <p>001 7:0 7:0 Tests CCINTCNTA</p> <p>010 15:8 15:8 Tests CCINTCNTB</p> <p>011 23:16 23:16 Tests CCINTCNTC</p> <p>100 23:16, 15:8, 7:0 23:16 All three sub-counters run in parallel</p> <p>101 none 7:0 Not used</p> <p>110 none 15:8 Tests carry from CCIntCntA to CCINTCNTB</p> <p>111 none 23:16 Tests carry from CCINTCNTB to CCINTCNTC</p>	RW	0
[27]	UNUSED_SPARE	Reserved	RW	0
[26:24]	TRPREFSYN[<p>Interval between REFP and subsequent REFA or ACT packets to the same bank.</p> <p>Largest tRP,REF spec of any installed RDRAM; tCYCLE granularity.</p> <p>010: t RPREF = 8 t CYCLE.</p> <p>011: t RPREF = 12 t CYCLE</p> <p>100: t RPREF = 16 t CYCLE</p> <p>...</p> <p>111: t RPREF = 28 t CYCLE</p>	RW	0
[23:20]	TRASREFSYN	<p>Interval between REFA and a subsequent REFP packet to the same bank. Largest tRAS,REF spec of any installed RDRAM; tCYCLE granularity.</p> <p>0100: t RASREF = 16 * t CYCLE</p> <p>0101: t RASREF = 20 * t CYCLE</p> <p>0110: t RASREF = 24 * t CYCLE</p> <p>...</p> <p>1111: t RASREF = 60* t CYCLE</p>	RW	0
[19]	STEP2_RAC_CC_EN	This bit needs to be asserted prior to doing step2 of the initialization process and should be de-asserted immediately after that. This causes the STOPTDx pins in the RAC to be deasserted during the initial current control operations. This is a necessary step for initialization.	RW	0
[18:16]	TOFFP	<p>Offset. Offset from a RDA or PREX packet to the equivalent PRER packet</p> <p>100: t OFFP = 4 t CYCLE</p> <p>101: t OFFP = 5 t CYCLE</p> <p>110: t OFFP = 6 t CYCLE</p> <p>111: t OFFP = 7 t CYCLE</p> <p>Other values are reserved</p>	RW	0



Bits	Field	Description	RW	Reset
[15:12]	TRP	Row Precharge. Largest tRP spec of any installed RDRAM; tCYCLE granularity. 0110: t RP = 6 t CYCLE 1000: t RP = 8 t CYCLE 1010: t RP = 10 t CYCLE 1100: t RP = 12 t CYCLE Other values are reserved.	RW	0
[11]	UNUSED_SPARE	Reserved	RW	0
[10:8]	TRASSYN	Largest tRAS spec of any installed RDRAM; tCYCLE granularity. 100: t RAS = 16 t CYCLE 101: t RAS = 20 t CYCLE 110: t RAS = 24 t CYCLE 111: t RAS = 28 t CYCLE Other values are reserved.	RW	0
[7:4]	TCAC	CAS Access Delay. Largest tCAC spec of any installed RDRAM; tCYCLE granularity. 1000: t CAC = 8 t CYCLE 1001: t CAC = 9 t CYCLE 1010: t CAC = 10 t CYCLE 1011: t CAC = 11 t CYCLE 1100: t CAC = 12 t CYCLE Other values are reserved.	RW	0
[3:0]	TRCD	RAS to CAS Delay. Largest tRCD spec of any installed RDRAM; tCYCLE granularity. 0101: t RCD = 5 t CYCLE 0111: t RCD = 7 t CYCLE 1001: t RCD = 9 t CYCLE 1011: t RCD = 11 t CYCLE Other values are reserved.	RW	0

5.3.8 RDRAM_CONFIG_2 (# = 0,1,2)

The # symbol in the register name signifies the DRAM Channel. This register holds configuration parameters for timing and other controls.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED			CCCNT									REFBN KBITS		REFCNT							MAX_DEV_ID								



Bits	Field	Description	RW	Reset
[31:27]	Reserved		RW	0
[26:19]	CCCNT	In Normal Mode (CC_TEST = 0), CCINTCNT[23:0] resets to 0 and increment every PClk cycle. When CINTCNT[23:16]= CCCNT, it returns to 0 and CCRDY is decremented. For the test modes (CC_TEST .NE. 0), other subfields of CCINTCNT can be compared with CCCNT, and CCINTCNT does not return to 0 when a match occurs	RW	0
[18:16]	REFBNKBITS	Refresh bank bits. 010: 4 refresh banks 011: 8 refresh banks 100: 16 refresh banks 101: 32 refresh banks	RW	0
[15:5]	REFCNT	REFCNT resets to 0 and increments every PClk cycle it reaches REFCNT, then it returns to 0 and REFRDY is decremented.	RW	0
[4:0]	MAX_DEV_ID	Largest Device ID for any RDRAM; Device IDs must be contiguous starting with 0 so CFGMAXDEVID equals the number of RDRAMs, minus 1. Tells RMC2 how many RDRAMs to perform current calibration on. Every CFGMAXDEVID + 1 RDRAM CC transactions, CRRDY is decremented to request a RAC CC cycle.	RW	0

5.3.9 RDRAM_CONFIG_3 (# = 0,1,2)

The # symbol in the register name signifies the DRAM Channel. This register holds configuration parameters for timing and other controls.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
UNUSED_SPARE														BANK_BITS		ROW_BITS		2KB_PAGE	SP_CORE	DEP_BNK	NO_DEV									

Bits	Field	Description	RW	Reset
[31]	RAC_POWERUP	Powerup RAC <ul style="list-style-type: none"> • 0: RAC is not powered • 1: RAC is powered Software must write this to a 1 either after or with writing RAC_Reset to a 1.	RW	0
[30]	RAC_RESET	Reset the RAC <ul style="list-style-type: none"> • 0: RAC is reset • 1: RAC is not reset Software must write this to a 1 either before or with writing RAC_Powerup to a 1, and then write to a 0 after 7 uS or after detecting Clock_Lock, whichever is sooner.	RW	0
[29]	DO_STEP_5_CCTL_SRCTL	Do initialization step 5 as defined in Rambus Specs. To repeat step 5, if desired write a 0 followed by a 1.	RW	0
[28]	DO_STEP_6	Do initialization step 6 as defined in Rambus Specs. To repeat step 6, if desired write a 0 followed by a 1.	RW	0
[27:24]	INIT_REFRESH/CCTL_COUNT	Number of refresh/current control operations done during init step 5 and 6. The number of operations is 16 times this number	RW	0
[23]	ALL_INITIALIZATION_DONE	Software must write this bit to 1 when all initialization has been completed.	RW	0
[22]	STEP_5_BUSY	Indicates that initialization step 5 is in progress.	RO	0
[21]	STEP_6_BUSY	Indicates that initialization step 6 is in progress.	RO	0
[20]	MASK_CURRENT_CONTROL	DFT -- Intel private Masks current controls <ul style="list-style-type: none"> • 0: Current Control operations happen normally • 1: Current Control operations are disabled 	RW	0
[19:13]	RAC_CCTL_IN	Intel private. Reset value—64 decimal (0x40)	RW	0x40
[12:3]	RAC_T44_TIMER	DFT -- Intel private.	RW	0
[2]	CLOCK_LOCK	Value of RAC Clock_Lock signal.	RW	0
[1]	DO_RAC_INIT	Writing a 1 to this bit during RDRAM Initialization starts the RAC init process. Once this bit is set to 1, it should not be set back to 0.	RW	0
[0]	INIT_IN_PROGRESS	Indicates the status of RAC Init during RDRAM Initialization. Once RAC Initialization has started, the RDRAM Initialization process should not proceed until this bit returns to 0. <ul style="list-style-type: none"> • 1: RAC init not done • 0: RAC init done 	RO	0

5.3.11 RDRAM_MISC_RAC_CONTROL

This register is used for initialization.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	
RQ0SEL				RQ1SEL				STOPRQ	STOPTQ	NAP	HOLD	DROWSY	RESERVED	HCLKN	SYNCM	PHSTALL	RESERVED						DTXOR	RESERVED	DTALLZ	DTIDDQ	BIMODE	SYNCLK_SYNC	BYPASS	RESERVED	

Bits	Field	Description	RW	Reset
[31:28]	RQ0SEL		RW	0
[27:24]	RQ1SEL		RW	0
[23]	STOPRQ	Inverted and sent to RAC	RW	0
[22]	STOPTQ		RW	0
[21]	NAP		RW	0
[20]	HOLD		RW	0
[19]	DROWSY		RW	0
[18]	RESERVED	Reserved	RW	0
[17]	HCLKN		RW	0
[16]	SYNCM		RW	0
[15]	PHSTALL	Moves SYNCLK by 90 degree phase	RW	0
[14:9]	RESERVED	Reserved	RW	0
[8]	DTXOR		RW	0
[7]	RESERVED	Reserved	RW	0
[6]	DTALLZ		RW	0
[5]	DTIDDQ		RW	0
[4]	BIMODE		RW	0
[3]	SYNCLK_SYNC	Inverted and sent to RAC	RW	0
[2]	BYPASS		RW	0
[1:0]	RESERVED	Reserved	RW	0

5.3.12 RDRAM_RAC_CONFIG

This register is private for Intel use and should not be written.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																																

Bits	Field	Description	RW	Reset
[31:0]	RESERVED	Reserved	RW	0

5.3.13 RDRAM_1066_CONFIG_GROUP (# = 0,1,2)

The # symbol in the register name signifies the DRAM Channel.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
VALUE																																

Bits	Field	Description	RW	Reset
[31:0]	VALUE	Each bit position represents the corresponding DevID value in the dram. For 1066MHz or higher operation, if '0' it means that RDRAM is leveled to return data at CfgTcac. If '1', it means that RDRAM is leveled to return data at CfgTcac +4*t CYCLE. The RMC inserts the necessary 4*t CYCLE bubbles between RD packets to prevent data from the two groups of RDRAMs overlapping on the Channel.	RW	0

5.3.14 RDRAM_SERIAL_CONFIG (# = 0,1,2)

The # symbol in the register name signifies the DRAM Channel. DRAM Clock is divided by RegCnt to generate SCK for all SIO transactions but Nap and Powerdown Exit, for which CrClk is divided by NapXCnt. (DRAM Clock is derived from via the CLOCK_CONTROL CSR) PdnX is the number of SCK cycles (divided by 256) CrBusy is asserted and SCK toggles following a Powerdown Exit command and must be greater than or equal to the value in the RDRAM PDNX register. NapX is the number of SCK cycles CrBusy and SCK toggles following a Nap Exit command and must be greater than or equal to the value in the RDRAM NAPX register. RIntStart is the number of CrClk cycles after CrStart is asserted for a Nap Exit before TQEn is de-asserted, to prevent the RMC transmitting ROW or COL packets during the restricted interval. RIntWidth is the number of CrClk cycles TQEn is de-asserted.

The value left in this register at the end of the initialization sequence should be acceptable for normal operation.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RINT_WIDTH		RINT_START				NAPX			PDNX			NAPXCNT		SCLK_FREQ_DIVISOR																		

Bits	Field	Description	RW	Reset
[31:28]	RINT_WIDTH	Do not overwrite these bits after initialization is complete.	RW	0
[27:20]	RINT_START		RW	0x00
[19:16]	NAPX		RW	0x1
[15:12]	PDNX		RW	0x1
[11:8]	NAPXCNT		RW	0x4
[7:0]	SCLK_FREQ_DIVISOR	(REGCNT) Divider value used to derive SCK from DRAM Clock Frequency.	RW	0x64

5.3.15 RDRAM_K0 through RDRAM_K11 (# = 0,1,2)

These registers are available in IXP2800 Rev B. The # symbol in the register name signifies the DRAM Channel. These registers must be initialized when three channels are enabled in RDRAM_CONTROL. When one or two channels are enabled, these registers are not used. The value loaded is based on the amount of memory populated on the channel as shown in [Table 5-10](#), [Table 5-11](#), and [Table 5-12](#).

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
Reserved										VALUE																					

Bits	Field	Description	RW	Reset
[31:23]	Reserved		RO	0
[22:0]	Constant Value	Value from Table 5-10 , Table 5-11 , or Table 5-12 .	RW	0

Table 5-10. RDRAM Constants (Hexadecimal) for 3-Channel Mode Part 1

MBytes	32	64	96	128	160	192	224	256
K11	NA	NA	NA	NA	NA	NA	NA	NA
K10	NA	NA	NA	NA	13FFFF	17FFFF	1BFFFF	1FFFFF
K9	NA	7FFFF	BFFFF	FFFFF	13FFFE	17FFFE	1BFFFE	1FFFFE
K8	3FFFF	7FFFE	BFFFD	FFFC	13FFFB	17FFFA	1BFFF9	1FFFF8
K7	3FFFC	7FFF8	BFFF4	FFF0	13FFEC	17FFE8	1BFFE4	1FFFE0
K6	3FFF0	7FFE0	BFFD0	FFC0	13FFB0	17FFA0	1BFF90	1FFF80
K5	3FFC0	7FF80	BFF40	FFF0	13FEC0	17FE80	1BFE40	1FFE00
K4	3FF00	7FE00	BFD00	FFC00	13FB00	17FA00	1BF900	1FF800
K3	3FC00	7F800	BF400	FF000	13EC00	17E800	1BE400	1FE000
K2	3F000	7E000	BD000	FC000	13B000	17A000	1B9000	1F8000
K1	3C000	78000	B4000	F0000	12C000	168000	1A4000	1E0000
K0	30000	60000	90000	C0000	F0000	120000	150000	180000

Table 5-11. RDRAM Constants (Hexadecimal) for 3-Channel Mode Part 2

MBytes	288	320	352	384	416	448	480	512
K11	NA							
K10	23FFFF	27FFFF	2BFFFF	2FFFFF	33FFFF	37FFFF	3BFFFF	3FFFFF
K9	23FFFD	27FFFD	2BFFFD	2FFFFD	33FFFC	37FFFC	3BFFFC	3FFFFC
K8	23FFF7	27FFF6	2BFFF5	2FFFF4	33FFF3	37FFF2	3BFFF1	3FFFF0
K7	23FFDC	27FFD8	2BFFD4	2FFFD0	33FFC8	37FFC8	3BFFC4	3FFFC0
K6	23FF70	27FF60	2BFF50	2FFF40	33FF30	37FF20	3BFF10	3FFF00
K5	23FDC0	27FD80	2BFD40	2FFD00	33FCC0	37FC80	3BFC40	3FFC00
K4	23F700	27F600	2BF500	2FF400	33F300	37F200	3BF100	3FF000
K3	23DC00	27D800	2BD400	2FD000	33CC00	37C800	3BC400	3FC000
K2	237000	276000	2B5000	2F4000	333000	372000	3B1000	3F0000
K1	21C000	258000	294000	2D0000	30C000	348000	384000	3C0000
K0	1B0000	1E0000	210000	240000	270000	2A0000	2D0000	300000

Table 5-12. RDRAM Constants (Hexadecimal) for 3-Channel Mode, Part 3

MBytes	544	576	608	640	672	704	736	768
K11	43FFFF	47FFFF	4BFFFF	4FFFFF	53FFFF	57FFFF	5BFFFF	5FFFFF
K10	43FFFE	47FFFE	4BFFFE	4FFFFE	53FFFE	57FFFE	5BFFFE	5FFFFE
K9	43FFFB	47FFFB	4BFFFB	4FFFB0	53FFFA	57FFFA	5BFFFA	5FFFA0
K8	43FFEF	47FFEE	4BFFED	4FFFE0	53FFEB	57FFEA	5BFFE9	5FFFE8
K7	43FFBC	47FFB8	4BFFB4	4FFFB0	53FFAC	57FFA8	5BFFA4	5FFFA0
K6	43FEF0	47FEE0	4BFED0	4FFEC0	53FEB0	57FEA0	5BFE90	5FFE80
K5	43FBC0	47FB80	4BFB40	4FFB00	53FAC0	57FA80	5BFA40	5FFA00
K4	43EF00	47EE00	4BED00	4FEC00	53EB00	57EA00	5BE900	5FE800
K3	43BC00	47B800	4BB400	4FB000	53AC00	57A800	5BA400	5FA000
K2	42F000	46E000	4AD000	4EC000	52B000	56A000	5A9000	5E8000
K1	3FC000	438000	474000	4B0000	4EC000	528000	564000	5A0000
K0	330000	360000	390000	3C0000	3F0000	420000	450000	480000

5.4 DDR SDRAM Controller - IXP2400

5.4.1 DDR SDRAM Register Map

The DDR registers are addressed at 8 byte offsets. Each register is 32 bits and data is transferred on the low 32 bits of the DRAM Push/Pull Data Busses.

Table 5-13. DDR SDRAM Register Map

Abbreviation	Offset	Name	Description
DU_CONTROL	0x000	DRAM Controller Control Register	Contains programmable delay/latency parameters to support various configurations
DU_ERROR_STATUS_1	0x008	DRAM Error Status Register 1	Logs the Address of transaction which had an ECC error
DU_ERROR_STATUS_2	0x010	DRAM Error Status Register 2	Logs details about type of ECC error
DU_ECC_TEST	0x018	DRAM ECC Test Register	Has control settings which can be used to inject false ECC errors for testing purposes
DU_INIT	0x020	DRAM Initialization Register	Contains controls for the DDR Mode register set, refresh, precharge commands
DU_CONTROL2	0x028	DRAM Controller Control Register 2	Contains additional DRAM Controller control fields
-	0x030 - 0x0F8	-	Reserved
DU_IO_CONFIG[1:224]	0x100 - 0x7F8	DRAM I/O Configuration Registers	Contains Drive strength controls for various interface pins. For Intel Internal use only.

5.4.2 DRAM Controller Control Register (DU_CONTROL)

The DRAM Controller Control Register contains programmable delay/latency parameters to support various DDR configurations. The NUM_CHANNELS and ENABLE_CNTLRL fields are ignored in IXP2400 because IXP2400 supports only 1 Channel.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X32_PART_SIZE	D_S_ICA_P	TRFC	TRRD	TWR	NUM_SIDES	NUM_ROW_COL	REF_COUNT						REF_EN	ENABLE_CNTLRL	NUM_CHANNELS	TWTR	RD_WR_SPACING	RD_RD_SPACING	TRC	TRAS	TCL	TRCD	TRP								

Bits	Field	Description	RW	Reset
[31]	X32_PART_SIZE	For IXP2400 Rev B only. Reserved otherwise. When the NUM_ROW_COL field contains the value 111, this bit specifies the size of the X32 parts used. 0 - 2M X 32 1 - 4M X 32	RO, RW for IXP2400 Rev B	0
[30]	DIS_CAP	For IXP2400 Rev B only. Reserved otherwise. Disable concurrent Auto Precharge 1 - Support DRAM parts that do not feature Concurrent Auto Precharge 0 - Use with DRAM parts that feature Concurrent Auto Precharge	RO, RW for IXP2400 Rev B	0
[29]	TRFC	Refresh Command Period <u>Encoding</u> 0 - 11 DRAM clocks 1 - 10 DRAM clocks	RW	0
[28]	TRRD	Active BankA to Active BankB delay <u>Encoding</u> 0 - 3 DRAM clocks 1 - 2 DRAM clocks	RW	0
[27]	TWR	Write recovery time <u>Encoding</u> 0 - 3 DRAM clocks 1 - 2 DRAM clocks	RW	0
[26]	NUM_SIDES	Number of DIMM Sides or physical memory banks. <u>Encoding</u> 0 - 1 side 1 - 2 sides	RW	0
[25:23]	NUM_ROW_COL	Indicates the number of device row and column bits by device type. <u>Encoding</u> 000 - 8M X 8 or 8M X 16 001 - 16M X 8 010 - 16M X 16 011 - 32M X 8 or 32M X 16 100 - 64M X 8 101 - 64M X 16 110 - 128M X 8 111 - X32 parts (IXP2400 Rev B only)	RW	X
[22:15]	REF_COUNT	Refresh counter reload value. At 300Mhz, the counter has a granularity of 213 ns and a maximum value of 54.61 us. If a double sided DIMM is used, the controller alternates auto-refreshes between the 2 sides. Therefore the refresh count should be set to half the desired interval in this case	RW	X
[14]	REF_EN	Enables Refresh operation. This bit should be set after loading the Refresh counter and setting the Refresh mode in DRAM to be Auto Refresh.	RW	0
[13]	ENABLE_CNTRLR	Not used in IXP2400	RO	0
[12:11]	NUM_CHANNELS	Not used in IXP2400	RO	00

Bits	Field	Description	RW	Reset
[10]	TWTR	Write command to Read command delay. This field controls the turnaround time on the DQ bus for WR-RD pairs to the same side. Encoding Turn-Around 0 2 DRAM clocks 1 1 DRAM clock	RW	0
[9]	RD_WR_SPACING	Back to Back Read-Write Turn Around. This field controls the turnaround time on the DQ bus for RD-WR pairs. Encoding Turn-Around 0 2 DRAM clocks 1 1 DRAM clock	RW	0
[8]	RD_RD_SPACING	Back to Back Read Turn Around. This field controls the turnaround time on the DQ bus for RD-RD sequence to different rows Encoding Turn-Around 0 2 DRAM clocks 1 1 DRAM clock	RW	0
[7:6]	TRC	Active to Active (same bank) delay(tRC) Encoding tRC 00 10 DRAM clocks 01 9 DRAM clocks 10 8 DRAM clocks 11 7 DRAM clocks	RW	00
[5:4]	TRAS	Activate to Precharge delay (tRAS). This bit controls the number of DRAM clocks for tRAS Encoding tRAS 00 7 DRAM clocks 01 6 DRAM clocks 10 5 DRAM clocks	RW	00



Bits	Field	Description	RW	Reset
[3:2]	TCL	CAS# Latency (tCL). This bit controls the number of clocks inserted between the registration of a READ command and the availability of the first piece of output data Encoding tCL 00 3 01 2.5 10 2 11 RESERVED	RW	00
[1]	TRCD	DRAM RAS# to CAS# Delay (tRCD). This bit controls the number of clocks inserted between a row activate command and a read or write command in that row Encoding tRCD 0 3 DRAM clocks 1 2 DRAM clocks	RW	0
[0]	TRP	DRAM RAS# Precharge (tRP). This bit controls the number of clocks that are inserted between a row precharge command and an activate command to the same row Encoding tRP 0 3 DRAM clocks 1 2 DRAM clocks	RW	0

5.4.3 DRAM Error Status Register 1 (DU_ERROR_STATUS_1)

The DRAM Error Status Register 1 records the type and address of a location with an ECC error. The Error Address field is invalid unless either the Uncorr_Error or Corr_Error bit is set. If the Corr_Error bit is set and the Uncorr_Error bit equals 0, then the Error Address field contains the address for the first correctable error that occurred. If the Uncorr_Error bit is set, then the Error Address field contains the address for the first uncorrectable error that occurred. This implies that if there are multiple correctable errors, the information of only the first error is preserved, and once an uncorrectable error occurs, the information of all subsequent errors is lost.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED	ERR_ADDR																									RESERVED	UNCORR_ERR	CORR_ERR			

Bits	Field	Description	RW	Reset
[31]	reserved		RO	0
[30:3]	ERR_ADDR	Address of the DRAM request which had an ECC error.	RO	X

Bits	Field	Description	RW	Reset
[2]	reserved		RO	0
[1]	UNCORR_ERR	Uncorrectable Error bit. If set, indicates that an uncorrectable error was detected. If this bit gets set, an Interrupt signal indicating uncorrectable error is asserted.	RW1C	0
[0]	CORR_ERR	Correctable Error bit. If set, indicates that a correctable error was detected. If this bit gets set, an Interrupt signal indicating correctable error is asserted.	RW1C	0

5.4.4 DRAM Error Status Register 2 (DU_ERROR_STATUS_2)

The DRAM Error Status Register 2 records the syndrome of an ECC error, and the originator of the read or RMW. The information in this register is valid only when either the Uncorr_Error or Corr_Error bit in the DU_Error_Status_1 register is set. If the Corr_Error bit is set and the Uncorr_Error bit equals 0, then this register contains the information for the first correctable error that occurred. If the Uncorr_Error bit is set, then this register contains the information for the first uncorrectable error that occurred. This implies that if there are multiple correctable errors, the information of only the first error is preserved, and once an uncorrectable error occurs, the information of all subsequent errors is lost.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
MULTIPLE_CORR_ERROR	MULTIPLE_UNCORR_ERROR	RMW_ERROR	RESERVED				ME				CONTEXT			SOURCE	RESERVED				ERR_SYND								

Bits	Field	Description	RW	Reset
[31]	Multiple Corr Error	Indicates that multiple correctable errors have occurred. This bit is reset when the Corr_Error bit in DU_Error_Status_1 is cleared	RO	0
[30]	Multiple Uncorr Error	Indicates that multiple uncorrectable errors have occurred. This bit is reset when the Uncorr_Error bit in DU_Error_Status_1 is cleared.	RO	0
[29]	RMW_ERROR	Indicates that the Error information captured was from an RMW operation	RO	X
[28:25]	reserved		RO	0
[24:20]	ME	Indicates which ME was the originator of the transaction that had an error. This field captures bits 31:27 of the Push/Pull ID. Only valid if Source field (bit 16) is a 1	RO	X

Bits	Field	Description	RW	Reset
[19:17]	CONTEXT	Indicates the originator of the transaction (based on the Source bit) that had an error. If Source is 1, this field is the ME Context number and this field captures bits 26:24 of the Push/Pull ID. If Source is 0 this field indicates Intel XScale® core or PCI as follows. 000 - Intel XScale® core 001 - PCI Others - Reserved	RO	X
[16]	SOURCE	Indicates, in conjunction with the Context and ME fields the originator of the transaction that had an error. 0 - Intel XScale® core or PCI 1 - ME	RO	X
[15:8]	reserved		RO	0
[7:0]	ERR_SYND	Syndromes bits generated by the ECC check logic. The Syndrome can be used to determine which bit had an error. Table 5-14	RO	X

Table 5-14. RR_SYND values and error bit position mapping

ERR_SYND	Bit	ERR_SYND	Bit	ERR_SYND	Bit	ERR_SYND	Bit
0x1C	0	0x2C	16	0x4C	32	0x8C	48
0x22	1	0xA4	17	0xA8	33	0xA1	49
0x51	2	0x52	18	0x54	34	0x58	50
0x0E	3	0xD0	19	0xF8	35	0x4F	51
0x94	4	0x98	20	0x0B	36	0x70	52
0x68	5	0x61	21	0x91	37	0x92	53
0x43	6	0x83	22	0x62	38	0x64	54
0xF1	7	0x2F	23	0x23	39	0x13	55
0xC1	8	0xC2	24	0xC4	40	0xC8	56
0x2A	8	0x4A	25	0x8A	41	0x1A	57
0x15	10	0x25	26	0x45	42	0x85	58
0xE0	11	0x0D	27	0x8F	43	0xF4	59
0x49	12	0x89	28	0xB0	44	0x07	60
0x86	13	0x16	29	0x19	45	0x29	61
0x34	14	0x38	30	0x26	46	0x46	62
0x1F	15	0xF2	31	0x32	47	0x31	63

5.4.5 DRAM ECC Test Register (DU_ECC_TEST)

The DRAM ECC Test Register has control settings which can be used to inject false ECC errors for testing purposes.



3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						DISABLE_CHK	ECC_INV								

Bits	Field	Description	RW	Reset
[31:9]	reserved		RO	0
[8]	DISABLE_CHK	Disable ECC checking bit. If set, disables checking for errors and any correction of data.	RW	0
[7:0]	ECC_INV	Bits provided by user (software) which are XORed with the H/W generated ECC bits. If any of these bits is "1", the corresponding ECC bit is inverted when data is written into DRAM. Does not affect reads. These bits should be all zero during normal operation.	RW	0

5.4.6 DRAM Initialization Register (DU_INIT)

The DRAM Initialization Register contains controls for the DRAM Mode Register set, refresh, and precharge commands. This register is used by the Initialization sequence to initialize the DRAM chips as per the manufacturer’s requirements. This is a pseudo register, a write to this register triggers an operation on the DRAM bus. The CAS latency value written into the DRAM Mode Register should be the same as the setting in the DU_CONTROL register. The Burst length should be set to 4 and the Burst type should be set to “sequential”. Minimum delays between special ops using this register are not enforced by hardware. Software is responsible for ensuring that minimum delays between these ops are met.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
LD_MODE_REG	REFRESH	PRECHARGE	RESERVED											CKE	SIDE1	SIDE0	BANK_SEL	MODE_BITS										

Bits	Field	Description	RW	Reset
[31]	LD_MODE_REG	Perform “Load Mode Register” operation to DRAM. A value of “00” on the BANK_SEL field selects the Mode Register and “01” selects the Extended Mode Register. Other values of BANK_SEL are reserved. This bit is cleared by hardware when the operation is complete.	RW1S	0
[30]	REFRESH	Perform an Auto Refresh operation to DRAM. This is used to do an explicit refresh operation during Initialization. This bit is cleared by hardware when the operation is complete.	RW1S	0
[29]	PRECHARGE	Perform a Precharge operation to DRAM. This is used to do an explicit refresh operation during Initialization. Set MODE_BITS[10] to 1 to precharge all banks. If MODE_BITS[10] is 0, the BANK_SEL field determines which bank is precharged. This bit is cleared by hardware when the operation is complete.	RW1S	0
[28:17]	reserved		RO	0
[16]	CKE	Clock Enable The value in this bit is driven to the D_CKE[1:0] pins.	RW1S	0
[15]	SIDE1	Apply the command specified by bits [31:29] to side 1 of a two sided DIMM. This bit controls the assertion of CS1#.	WO	X

Bits	Field	Description	RW	Reset
[14]	SIDE0	Apply the command specified by bits [31:29] to side 0 of the DIMM. For a one sided DIMM, this bit should be set to 1. This bit controls the assertion of CS0#.	WO	X
[13:12]	BANK_SEL	These bits select which internal bank is targeted by the command specified in bits [31:29]. The BANK_SEL field corresponds to the BA[1:0] pins on the DRAM interface.	WO	X
[11:0]	MODE_BITS	Mode bits written into the DRAM when a "Load Mode Register" or "Load Enhanced Mode Register" command is issued. Refer to the datasheet for the DRAM part for the encoding of this field. MODE_BITS[10] is used to control whether a Precharge command targets all banks or a specific bank.	WO	X

5.4.7 DRAM Controller Control Register 2 (DU_CONTROL2)

The DRAM Controller Control Register 2 contains additional control fields that affect the operation of the DRAM Controller. These fields should be set during initialization of the controller and should not be changed during normal operation.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																		RCVEN_OV	PHASSEL[1:0]	RCVEN_DLY	RD_SKIP	WR_SKIP								

Bits	Field	Description	RW	Reset
[31:12]	reserved		RO	0
[11]	RCVEN_OV	Receive Enable Override Override CAS Latency setting to control RCVEN independently. If 1, PHASSEL[1:0] and RCVEN_DLY control the behavior of RCVEN. If 0, the CAS Latency setting controls RCVEN.	RW	0
[10:9]	PHASSEL[1:0]	Receive Enable Phase Select Coarse adjustment (.25 Dram Clk granularity) of RCVEN delay, if RCVEN_OV = 1. A value of 00 is no delay.	RW	00

Bits	Field	Description	RW	Reset
[8]	RCVEN_DLY	Receive Enable Delay Controls the timing of RCVEN signal. A value of 1 delays RCVEN by 1 Dram Clk, if RCVEN_OV = 1. A value of 0 is no delay.	RW	0
[7:4]	RD_SKIP	Read Skip Threshold Determines the number of times waiting read requests are skipped over by the Round Robin Scheduler in favor of waiting write requests before the Scheduler starts giving reads higher priority than writes. Permissible range of values: 4'b011 to 4'b1100 (3 to 12) Any other values will lead to unpredictable results.	RW	4'b0011
[3:0]	WRT_SKIP	Write Skip Threshold Determines the number of times waiting write requests are skipped over by the Round Robin Scheduler in favor of waiting read requests before the Scheduler starts giving writes higher priority than reads. Permissible range of values: 4'b011 to 4'b1100 (3 to 12) Any other values will lead to unpredictable results.	RW	4'b0011

5.4.8 DRAM RCOMP & I/O Registers

The DRAM RCOMP Registers contain drive strength controls for the interface pins. This is needed in order to support different board configurations: On board DRAM/single sided DIMM/double sided DIMM. These registers should be set to an appropriate value during initialization, before the DDR is used. All of these registers are R/W. Most of the fields are not reset by system reset, so are undefined until they are written. The reset value of fields which are reset is indicated in the field description. Where less than 32 bits are used, the lower bits are used. Any unused bits are undefined when read, and should be written with 0 unless indicated otherwise.

Table 5-15. DRAM RCOMP & I/O Configuration Register Map

Abbreviation	Offset	Bits Used	Description	Reset
cr0_frscmrcomp	0x100	[0]	Force an SM RCOMP	0
cr0_rcompprd	0x108	[2:0]	MCH R Comp period 000 - once every 2097152 DDR clock cycles 001 - once every 1048576 DDR clock cycles 010 - once every 524288 DDR clock cycles 011 - once every 262144 DDR clock cycles 100 - once every 131072 DDR clock cycles 101 - once every 65536 DDR clock cycles 110 - once every 32768 DDR clock cycles 111 - no rcomp updates	000
cr0_blkrcomp	0x110	[0]	Block the RCOMP FSM	0
cr0_digfil	0x118	[1:0]	Digital Filtering Select 00 - No Filtering 01 - Filter Clamp Value = 2 10 - Filter Clamp Value = 4 11 - Filter Clamp Value = 16	01

Table 5-15. DRAM RCOMP & I/O Configuration Register Map

cr0_rcompbit7_chk	0x120	[0]	Indicates to consider whether rcomp bit 7 is set for slew table indexing	0
cr0_slewprogrammed	0x128	[0]	Indicates when slew tables are programmed. This bit is set by software to indicate when initialization is done.	0
cr0_dstrengthsel	0x130	[26:24]	Strength Select for cckx16 Strength select 3 bit encoding for all the fields: 000 = .75x 001 = 1x 010 = 1.25x 011 = 1.50x 100 = 2x 101 = 2.5x 110 = 3x 111 = 4x	000
		[23:21]	Strength Select for cckx8	000
		[20:18]	Strength Select for csx16	000
		[17:15]	Strength Select for csx8	000
		[14:12]	Strength Select for ckex16	000
		[11:9]	Strength Select for ckex8	000
		[8:6]	Strength Select for rcv	000
		[5:3]	Strength Select for ctl	000
		[2:0]	Strength Select for dq DQ pin strength is decided by the jt_config register, if jt_config[21] is "1"	000
cr0_ovrrideh	0x138	[17:0]	Override for Horizontal P and N. 17 = Override select for N. Reset value is 1. 16:9 = N override value. Reset value is 0x80. 8 = Override select for P. Reset value is 1. 7:0 = P override value. Reset value is 0x80.	0x308 0
cr0_ovrridev	0x140	[17:0]	Override for Vertical P and N 17 = Override select for N. Reset value is 1. 16:9 = N override value. Reset value is 0x80. 8 = Override select for P. Reset value is 1. 7:0 = P override value. Reset value is 0x80.	0x308 0
cr0_ddqrcmp	0x148	[15:0]	DDR DQ/DQS Rcomp offset register 15 = Sign for pulldown 14:8 = Value for pulldown 7 = Sign for pullup 6:0 = Value for pullup	
cr0_ddqpslew0	0x150	[31:0]	DDR DQ/DQS pull-up Slew lookup table register 0	
cr0_ddqpslew1	0x158	[31:0]	DDR DQ/DQS pull-up Slew lookup table register 1	
cr0_ddqpslew2	0x160	[31:0]	DDR DQ/DQS pull-up Slew lookup table register 2	
cr0_ddqpslew3	0x168	[31:0]	DDR DQ/DQS pull-up Slew lookup table register 3	
cr0_ddqnslew0	0x170	[31:0]	DDR DQ/DQS pull-dn Slew lookup table register 0	
cr0_ddqnslew1	0x178	[31:0]	DDR DQ/DQS pull-dn Slew lookup table register 1	
cr0_ddqnslew2	0x180	[31:0]	DDR DQ/DQS pull-dn Slew lookup table register 2	

Table 5-15. DRAM RCOMP & I/O Configuration Register Map

cr0_ddqnslew3	0x188	[31:0]	DDR DQ/DQS pull-dn Slew lookup table register 3	
cr0_dctlrcomp	0x190	[15:0]	MA, BA, RAS#, CAS#, WE# Rcomp offset register 15 = Sign for pulldown 14:8 = Value for pulldown 7 = Sign for pullup 6:0 = Value for pullup	
cr0_dctlpslew0	0x198	[31:0]	MA, BA, RAS#, CAS#, WE# pull-up Slew lookup table register 0	
cr0_dctlpslew1	0x1A0	[31:0]	MA, BA, RAS#, CAS#, WE# pull-up Slew lookup table register 1	
cr0_dctlpslew2	0x1A8	[31:0]	MA, BA, RAS#, CAS#, WE# pull-up Slew lookup table register 2	
cr0_dctlpslew3	0x1B0	[31:0]	MA, BA, RAS#, CAS#, WE# pull-up Slew lookup table register 3	
cr0_dctlnslew0	0x1B8	[31:0]	MA, BA, RAS#, CAS#, WE# pull-dn Slew lookup table register 0	
cr0_dctlnslew1	0x1C0	[31:0]	MA, BA, RAS#, CAS#, WE# pull-dn Slew lookup table register 1	
cr0_dctlnslew2	0x1C8	[31:0]	MA, BA, RAS#, CAS#, WE# pull-dn Slew lookup table register 2	
cr0_dctlnslew3	0x1D0	[31:0]	MA, BA, RAS#, CAS#, WE# pull-dn Slew lookup table register 3	
cr0_drcvrcomp	0x1D8	[15:0]	RCV Rcomp offset register 15 = Sign for pulldown 14:8 = Value for pulldown 7 = Sign for pullup 6:0 = Value for pullup	
cr0_drcvpslew0	0x1E0	[31:0]	RCV pull-up Slew lookup table register 0	
cr0_drcvpslew1	0x1E8	[31:0]	RCV pull-up Slew lookup table register 1	
cr0_drcvpslew2	0x1F0	[31:0]	RCV pull-up Slew lookup table register 2	
cr0_drcvpslew3	0x1F8	[31:0]	RCV pull-up Slew lookup table register 3	
cr0_drcvnslew0	0x200	[31:0]	RCV pull-dn Slew lookup table register 0	
cr0_drcvnslew1	0x208	[31:0]	RCV pull-dn Slew lookup table register 1	
cr0_drcvnslew2	0x210	[31:0]	RCV pull-dn Slew lookup table register 2	
cr0_drcvnslew3	0x218	[31:0]	RCV pull-dn Slew lookup table register 3	
cr0_dx8x16ckeckscksel	0x220	[5:0]	Selects one of two possible slew compensation values for all CKEs, CSs, and CKs. 0 - indicates x16 1 - indicates x8	0x00
cr0_dckercomp	0x228	[31:0]	CKE Rcomp offset register 31:16 = Pullup/down offset sign/value for x16 15:0 = Pullup/down offset sign/value for x8	
cr0_dckex8pslew0	0x230	[31:0]	CKE x8 pull-up Slew lookup table register 0	
cr0_dckex8pslew1	0x238	[31:0]	CKE x8 pull-up Slew lookup table register 1	
cr0_dckex8pslew2	0x240	[31:0]	CKE x8 pull-up Slew lookup table register 2	

Table 5-15. DRAM RCOMP & I/O Configuration Register Map

cr0_dckex8pslew3	0x248	[31:0]	CKE x8 pull-up Slew lookup table register 3	
cr0_dckex8nslew0	0x250	[31:0]	CKE x8 pull-dn Slew lookup table register 0	
cr0_dckex8nslew1	0x258	[31:0]	CKE x8 pull-dn Slew lookup table register 1	
cr0_dckex8nslew2	0x260	[31:0]	CKE x8 pull-dn Slew lookup table register 2	
cr0_dckex8nslew3	0x268	[31:0]	CKE x8pull-dn Slew lookup table register 3	
cr0_dckex16pslew0	0x270	[31:0]	CKE x16 pull-up Slew lookup table register 0	
cr0_dckex16pslew1	0x278	[31:0]	CKE x16 pull-up Slew lookup table register 1	
cr0_dckex16pslew2	0x280	[31:0]	CKE x16 pull-up Slew lookup table register 2	
cr0_dckex16pslew3	0x288	[31:0]	CKE x16 pull-up Slew lookup table register 3	
cr0_dckex16nslew0	0x290	[31:0]	CKE x16 pull-dn Slew lookup table register 0	
cr0_dckex16nslew1	0x298	[31:0]	CKE x16 pull-dn Slew lookup table register 1	
cr0_dckex16nslew2	0x2A0	[31:0]	CKE x16 pull-dn Slew lookup table register 2	
cr0_dckex16nslew3	0x2A8	[31:0]	CKE x16 pull-dn Slew lookup table register 3	
cr0_dcsrcomp	0x2B0	[31:0]	CS# Rcomp offset register 31:16 = Pullup/down offset sign/value for x16 15:0 = Pullup/down offset sign/value for x8	
cr0_dcsx8pslew0	0x2B8	[31:0]	CS# x8 pull-up Slew lookup table register 0	
cr0_dcsx8pslew1	0x2C0	[31:0]	CS# x8 pull-up Slew lookup table register 1	
cr0_dcsx8pslew2	0x2C8	[31:0]	CS# x8 pull-up Slew lookup table register 2	
cr0_dcsx8pslew3	0x2D0	[31:0]	CS# x8 pull-up Slew lookup table register 3	
cr0_dcsx8nslew0	0x2D8	[31:0]	CS# x8 pull-dn Slew lookup table register 0	
cr0_dcsx8nslew1	0x2E0	[31:0]	CS# x8 pull-dn Slew lookup table register 1	
cr0_dcsx8nslew2	0x2E8	[31:0]	CS# x8 pull-dn Slew lookup table register 2	
cr0_dcsx8nslew3	0x2F0	[31:0]	CS# x8pull-dn Slew lookup table register 3	
cr0_dcsx16pslew0	0x2F8	[31:0]	CS# x16 pull-up Slew lookup table register 0	
cr0_dcsx16pslew1	0x300	[31:0]	CS# x16 pull-up Slew lookup table register 1	
cr0_dcsx16pslew2	0x308	[31:0]	CS# x16 pull-up Slew lookup table register 2	
cr0_dcsx16pslew3	0x310	[31:0]	CS# x16 pull-up Slew lookup table register 3	
cr0_dcsx16nslew0	0x318	[31:0]	CS# x16 pull-dn Slew lookup table register 0	
cr0_dcsx16nslew1	0x320	[31:0]	CS# x16 pull-dn Slew lookup table register 1	
cr0_dcsx16nslew2	0x328	[31:0]	CS# x16 pull-dn Slew lookup table register 2	
cr0_dcsx16nslew3	0x330	[31:0]	CS# x16pull-dn Slew lookup table register 3	
cr0_dckrcomp	0x338	[31:0]	CK CK# Rcomp offset register 31:16 = Pullup/down offset sign/value for x16 15:0 = Pullup/down offset sign/value for x8	
cr0_dckx8pslew0	0x340	[31:0]	CK CK# x8 pull-up Slew lookup table register 0	
cr0_dckx8pslew1	0x348	[31:0]	CK CK# x8 pull-up Slew lookup table register 1	
cr0_dckx8pslew2	0x350	[31:0]	CK CK# x8 pull-up Slew lookup table register 2	
cr0_dckx8pslew3	0x358	[31:0]	CK CK# x8 pull-up Slew lookup table register 3	
cr0_dckx8nslew0	0x360	[31:0]	CK CK# x8 pull-dn Slew lookup table register 0	

Table 5-15. DRAM RCOMP & I/O Configuration Register Map

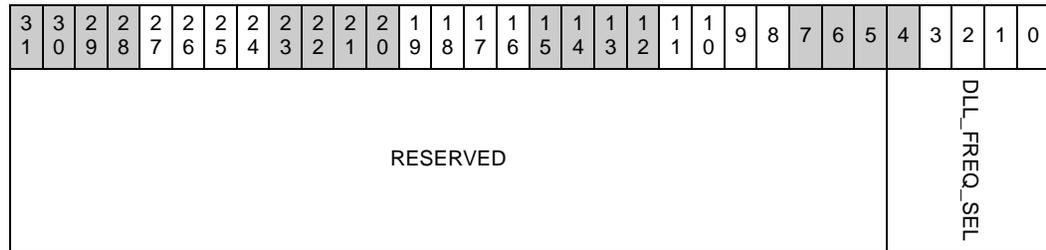
cr0_dckx8nslew1	0x368	[31:0]	CK CK# x8 pull-dn Slew lookup table register 1	
cr0_dckx8nslew2	0x370	[31:0]	CK CK# x8 pull-dn Slew lookup table register 2	
cr0_dckx8nslew3	0x378	[31:0]	CK CK# x8 pull-dn Slew lookup table register 3	
cr0_dckx16pslew0	0x380	[31:0]	CK CK# x16 pull-up Slew lookup table register 0	
cr0_dckx16pslew1	0x388	[31:0]	CK CK# x16 pull-up Slew lookup table register 1	
cr0_dckx16pslew2	0x390	[31:0]	CK CK# x16 pull-up Slew lookup table register 2	
cr0_dckx16pslew3	0x398	[31:0]	CK CK# x16 pull-up Slew lookup table register 3	
cr0_dckx16nslew0	0x3A0	[31:0]	CK CK# x16 pull-dn Slew lookup table register 0	
cr0_dckx16nslew1	0x3A8	[31:0]	CK CK# x16 pull-dn Slew lookup table register 1	
cr0_dckx16nslew2	0x3B0	[31:0]	CK CK# x16 pull-dn Slew lookup table register 2	
cr0_dckx16nslew3	0x3B8	[31:0]	CK CK# x16 pull-dn Slew lookup table register 3	
cr0_jt_config	0x3c0	[21]	Select Test mode Strength	0
		[20:18]	Test mode Strength	100
		[17]	Fast Reset Strap	0
		[16:9]	NRCOMP for test mode	0x90
		[8:1]	PRCOMP for test mode	0x90
		[0]	Test Mode Override	0
DDR_Rx_DLL	0x650	[4:0]	Selects Rx DLL tap	
DDR_Rx_Deskew	0x688	[4:0]	Sets the DLL to match the freq the DDR unit is programmed to run.	
DDR_RDDLYSEL_RECVEN	0x3C8	[4:0]	Selects the DLL tap used for delaying the Rcvenin strobe A related CSR is the DDR_Rx_DLL CSR.	

5.4.8.1 DDR_Rx_DLL

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																												DLL_TAP_SEL			

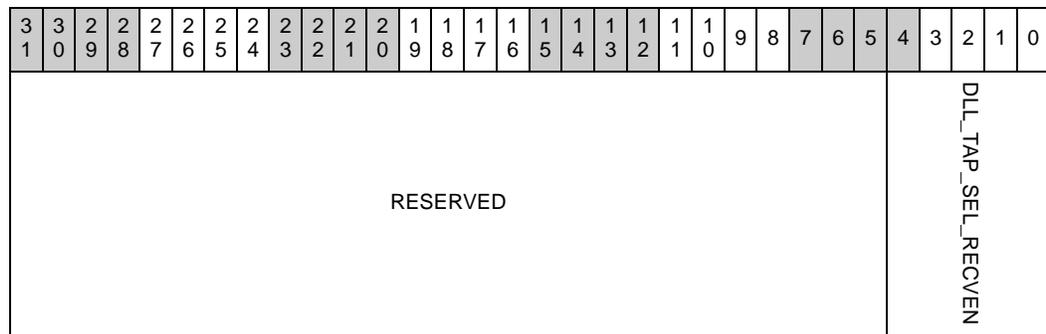
Bits	Field	Description	RW	Reset
[31:5]	Reserved		RO	
[4:0]	DLL_Tap_Sel	Selects the Rx DLL tap (except for Receive Enable signal) The Receive Enable signal tap is selected by the DDR_RDDLYSEL_RECVEN CSR	RW	00010

5.4.8.2 DDR_Rx_Deskew



Bits	Field	Description	RW	Reset
[31:5]	Reserved		RO	
[4:0]	DLL_Freq_Sel	Sets the DLL to match the freq the DDR unit is programmed to run.	RW	00011

5.4.8.3 DDR_RDDLSEL_RECVEN



Bits	Field	Description	RW	Reset
[31:5]	Reserved		RO	
[4:0]	DLL_Tap_Sel_Recven	Selects the DLL tap used for delaying the Rcvenin strobe. A related CSR is the DDR_Rx_DLL CSR.	RW	00010

5.5 SRAM QDR Controller

Table 5-16 shows the offset addresses of the SRAM Controller registers. Refer to [Chapter 4, “Address Maps”](#) for the base address and details on how they are accessed. These CSRs can be accessed by the Intel XScale® core, PCI and the MEs.

The IXP2800 supports four SRAM Channels. The IXP2400 supports two SRAM Channels, i.e. Channel 0 and Channel 1. Each channel supports their own register set.

Table 5-16. SRAM Register Summary (Sheet 1 of 2)

CSR name	Offset	Description	Section
SRAM_CONTROL	0x0000	SRAM controller configuration	Section 5.5.1
SRAM_PARITY_STATUS_1	0x0004	Parity control and recording of last faulty address	Section 5.5.2
SRAM_PARITY_STATUS_2	0x0008	Record of the source of request which generated parity error	Section 5.5.3
SPARE	0x000C	Reserved	Section 5.5.4
QDR_INTERNAL_PIPELINE	0x0108		Section 5.5.5
QDR_RX_DLL	0x0228		Section 5.5.6
QDR_RD_PTR_OFFSET	0x0240		Section 5.5.8
QDR_RX_DESKEW	0x0244		Section 5.5.7

Table 5-16. SRAM Register Summary (Sheet 2 of 2)

CSR name	Offset	Description	Section
Q_RCMP_SETUP_CONTROL	0x0300	All QDR System Memory RCOMP and Slew Rate control functions are consolidated into a memory mapped address region. It is expected that this address space will only be enabled temporarily by System BIOS, and then disabled to hide it from the Operating System.	Section 5.5.9.1
Q_RCMP_PMOS_MEASURED	0x0304		Section 5.5.9.2
Q_RCMP_NMOS_MEASURED	0x0308		Section 5.5.9.3
Q_RCMP_PMOS_OVERRIDE	0x030C		Section 5.5.9.4
Q_RCMP_NMOS_OVERRIDE	0x0310		Section 5.5.9.5
Q_RCMP_PMOS_NMOS_SCOMP_OVERRIDE	0x0314		Section 5.5.9.6
Q_RCMP_STRENGTH_SLEW_INDEX_SEL	0x0318		Section 5.5.9.8
Q_RCMP_ADDR_PMOS_PU_OFFSET	0x031C		Section 5.5.9.9
Q_RCMP_ADDR_NMOS_PD_OFFSET	0x0320		Section 5.5.9.10
Q_RCMP_DATA_PMOS_PU_OFFSET	0x0324		Section 5.5.9.11
Q_RCMP_DATA_NMOS_PD_OFFSET	0x0328		Section 5.5.9.12
Q_RCMP_KCLK_PMOS_PU_OFFSET	0x032C		Section 5.5.9.13
Q_RCMP_KCLK_NMOS_PD_OFFSET	0x0330		Section 5.5.9.14
Q_RCMP_DQ_PMOS_PU_OFFSET	0x0334		Section 5.5.9.15
Q_RCMP_DQ_NMOS_PD_OFFSET	0x0338		Section 5.5.9.16
Q_RCMP_PMOS_NMOS_VERT_OVERRIDE	0x033C		Section 5.5.9.17
Q_RCMP_ADDR_PMOS_PU_SLEW_TABLE_# (# = 0,1,2,3)	0x0340-0x034C		Slew Look Up Table Registers Section 5.5.9.18
Q_RCMP_ADDR_NMOS_PD_SLEW_TABLE_# (# = 0,1,2,3)	0x0350-0x035C		
Q_RCMP_DATA_PMOS_PU_SLEW_TABLE_# (# = 0,1,2,3)	0x0360-0x036C		
Q_RCMP_DATA_NMOS_PD_SLEW_TABLE_# (# = 0,1,2,3)	0x0370-0x037C		
Q_RCMP_KCLK_PMOS_PU_SLEW_TABLE_# (# = 0,1,2,3)	0x0380-0x038C		
Q_RCMP_KCLK_NMOS_PD_SLEW_TABLE_# (# = 0,1,2,3)	0x0390-0x039C		
Q_RCMP_DQ_PMOS_PU_SLEW_TABLE_# (# = 0,1,2,3)	0x03A0-0x03AC		
Q_RCMP_DQ_NMOS_PD_SLEW_TABLE_# (# = 0,1,2,3)	0x03B0-0x03BC		

5.5.1 SRAM_CONTROL

SRAM_CONTROL is used for static setup parameters.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED													QDR_SWIZZLE	Q_CTL_MODE	ENQUEUE_PERFORMANCE_MODE	QC_IGN_EOP	QC_IGN_SEG_CNT	PIPELINE	SRAM_SIZE	UNUSED_SPARE	PORT_CTL	PAR_EN	RESERVED						

Bits	Field	Description	RW	Reset
[31:18]	RESERVED	Reserved	RO	0
[17]	QDR_SWIZZLE	Swizzle the QDR read data bits: 0 - No change 1 - Swap QDR read data [15:0] with bits [31:16]	RW	0
[16]	Q_CTL_MODE	Reserved for future	RW	0
[15]	ENQUEUE_PERFORMANCE_MODE	For IXP2400 and IXP2800 Rev A: Setting this bit forces the Queue Controller to serialize enqueues and dequeues to the same queue. Clearing this bits allows the Queue Controller, to process enqueues and dequeues to the same queue in parallel, which improves the performance. For IXP2400 and IXP2800 Rev B: Reserved	RW	0
[14]	QC_IGN_EOP	Queue Controller Ignore EOP 1—always decrement Q_count on buffer dequeue without regard to the buffer EOP bit. 0—decrement Q_count on buffer dequeue only if buffer EOP is set Refer to Table 5-17 for a description of queuing modes	RW	0
[13]	QC_IGN_SEG_CNT	Queue Controller Ignore Segment Count 1—always do a full buffer dequeue and return the Segment Count field to the ME unchanged. 0—if buffer Segment Count is zero, then do a full buffer dequeue, else do a Segment dequeue and decrement buffer Segment Count Refer to Table 5-17 for a description of queuing modes	RW	0

Bits	Field	Description	RW	Reset
[12:10]	PIPELINE	Indicates the number of external pipeline delays to wait from read address driven from the IXP2800 / IXP2400 until registering the read data on the IXP2800 / IXP2400. Value of 0 is used when there are no external pipeline registers. 0—0 cycles (no external pipeline registers) 1—1 cycle 2—2 cycles 3—3 cycles 4—4 cycles 5, 6 and 7 are reserved values	RW	0
[9:7]	SRAM_SIZE	Indicates the size of each SRAM chip. This controls which address bits control when the port enables assert. Note—all the SRAM chips on a given channel must be the same size. (Note—Addresses listed here are the program generated address, not the SRAM address pins.) 000—512KB x 18 (1MB) —address [21:20] 001—1MB x 18 (2MB) —address [22:21] 010—2MB x 18 (4MB) —address [23:22] 011—4MB x 18 (8MB) —address [24:23] 100—8MB x 18 (16MB) —address [25:24] 101—16MB x 18 (32MB)—address [25] (only 2 SRAMs of this size can be used due to 64MB channel address capacity). 110—32MB x 18 (64MB)—none (only 1 SRAM of this size can be used due to 64MB channel address capacity). 111—external port enable decode—RPE_L[0] and WPE_L[0] assert for all accesses regardless of address. RPE_L[0]/WPE_L[0] are asserted for the lowest SRAM addresses, RPE_L[1]/WPE_L[1] are asserted for the next highest, etc.	RW	0
[6]	UNUSED_SPARE	Reserved	RW	0
[5:4]	PORT_CTL	Bit 5 enables SRAM Controller address [23:22] to be used as RPE_L[2]/WPE_L[2]. Bit 4 enables SRAM Controller address [21:20] to be used as RPE_L[3]/WPE_L[3]. 0—Use as addresses. 1—Use as Port Enables. These bits cannot be set for 8MB, 16MB, 32MB or 64M SRAMs because the address outputs are needed as addresses.	RW	3
[3]	PAR_EN	Indicates if the array is protected by Parity. 0—No Parity. 1—Parity.	RW	0
[2:0]	RESERVED	Reserved	RO	0

Table 5-17. Queueing Modes

Mode	Mode of Operation	SRAM_CONTROL[]	
		QC_IGN_EOP	QC_IGN_SEG_CNT
0	dequeue segments & count packets	0	0
1	dequeue buffers & count packets	0	1
2	Not valid	1	0
3	dequeue buffers & count buffers	1	1

5.5.2 SRAM_PARITY_STATUS_1

The SRAM_PARITY_STATUS_1 Register is used to record the address of parity errors, and allow incorrect parity to be written into the SRAM for test and diagnostic purposes.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0																										
WW_PAR		RESERVE		D																											ADDRESS																										

Bits	Field	Description	RW	Reset
[31:28]	WW_PAR	Write Wrong Parity. Enables test software to test the parity bits of the RAMs and checking logic. Bit 28 controls byte 0, bit 29 controls byte 1, etc. 0—write correct parity. 1—write incorrect parity.	RW	0
[27:24]	RESERVED	Reserved	RO	0
[23:0]	ADDRESS	Records the address which has a parity error. Only valid when one or more bits in SRAM_Parity[Error] is a 1.	RO	undef

5.5.3 SRAM_PARITY_STATUS_2

The SRAM_PARITY_STATUS_2 Register is used to record the originator of the read with parity error(s).

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
MULT_ERR	RESERVED				ME				THD				SRC	RESERVED										ERR							

Bits	Field	Description	RW	Reset
[31]	MULT_ERR	Multiple Errors. Indicates that another error occurred when one or more Error bits were already set.	RO	0
[30:25]	RESERVED	Read only as 0.	RO	undef
[24:20]	ME	Indicates which Microengine was the originator of the transaction that had an error. Only valid if Source is a 1.	RO	undef
[19:17]	THD	Thread. Indicates the originator of the transaction (based on the Source bit) that had an error. If Source is 1, this field is the thread number, if Source is 0 this field is the Intel XScale® core or HL as follows. 000—Intel XScale® core 001—PCI Others—Reserved	RO	undef
[16]	SRC	Source. Indicates, in conjunction with the Thread field the originator of the transaction (either a read or partial write that caused a read-modify-write) that had an error. 0—Intel XScale® core or PCI 1—Microengine	RO	undef
[15:4]	RESERVED	Reserved		undef
[3:0]	ERR	ERROR. Indicates that incorrect parity was detected on a read. Bit 0 indicates byte 0, etc. These bits will interrupt the Intel XScale® core if enabled in FIQ_ENABLE or IRQ_ENABLE. When any of these bits is set the Address field of SRAM_PARITY_STATUS_1 and the Source/Microengine/Thread fields of this register are frozen, which implies that the location of subsequent parity errors will be lost.	W1C	0

5.5.4 SPARE

SPARE is a test registers for Intel use only.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
SPARE																																

Bits	Field	Description	RW	Reset
[31:0]	SPARE	Reserved	RW	0x0ACE FACE

5.5.5 QDR_INTERNAL_PIPELINE

This register is for test, for use by Intel only.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																											PIPELINE			

Bits	Field	Description	RW	Reset
31:2	RESERVED	Reserved	RO	
1:0	PIPELINE	Adds or removes one cycle of internal read latency: 00 - Remove 1 cycle 01 - Nominal 10 - Add 1 cycle 11 - Reserved	RW	01

5.5.6 QDR_RX_DLL

The purpose of this register and the QDR_RX_DESKEW register is to adjust the receiving clock delay so that the receiving clock rising edge is located at the center of receiving data.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																											ACIOLB_DELAY	DLL_SLAVE_SEL		

Bits	Field	Description	RW	IXP2400 Reset	IXP2800 Reset
31:7	RESERVED	Reserved	RO	0	0
[6:5]	OnDie Termination (for IXP2400) ACIOLB_DELAY (for IXP2800)	For IXP2400, bit[6] enables OnDie Termination. Set to 1 for Enable and 0 for Disable. bit [5] is unused. For IXP2800, Intel Use only and must always be set to 0	RW	0	0
4:0	DLL_SLAVE_SEL	QDR DLL slave setting selects the number of slave taps in the DLL. The number of slave taps selected is approximately proportional to the DLL delay. The valid range is 0x0 to 0x17. This setting should be the same as DLL delay master setting (QDR_RX_DESKEW[4:0]). The recommended value from simulation is 0xA.	RW	0x2	0

5.5.7 QDR_RX_DESKEW

The purpose of this register and the QDR_RX_DLL register is to adjust the receiving clock delay so that the receiving clock rising edge is located at the center of receiving data.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																									DLL_BYPASS	DLL_MASTER						

Bits	Field	Description	RW	IXP2400 Reset	IXP2800 Reset
31:6	RESERVED	Reserved	RO	0	0
5	DLL_BYPASS	For Intel Use only and must always be set to 0	RW	0	0
4:0	DLL_MASTER	QDR DLL master setting selects the number of master taps in the DLL. The number of master taps selected is approximately inverse proportional to DLL delay. The valid range from simulation is 0x0 to 0xa. The recommended value from simulation is 0xa. The valid range and recommended value may change after silicon characterization.	RW	0x3	0

5.5.8 QDR_RD_PTR_OFFSET

This register is needed for configuration of QDR I Vs. QDR II interface.

This CSR controls which bank (odd or even bank) gets the first chunk of data.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																									Rx_Descramble	Rd_Wr_Ptr_Offset_Sel						

Bits	Field	Description	RW	Reset
31:3	Reserved	Reserved	RO	0
2	Rx_Descramble	Adjusts by 1/2 clock incrementing of the read pointer to the receiver's deskew FIFO. Affects by 1/2 clock what two groups of input data are selected to form a complete word when reading. On the IXP2400, the default value 0 is for QDR I. Set it to 1 for QDR II.	RW	0
1:0	Rd_Wr_Ptr_Offset_Sel	Selects the offset between the write & read pointers in the receiver's deskew FIFO. Affects the amount of external latency cycles available.	RW	0

5.5.9 QDR RCOMP Registers

These registers are used to set the drive strength for the QDR interface pins. The drive strength may need to be adjusted for a specific PC Board design and the number of QDR devices installed on a board. The setup procedure that is recommended for new board designs is described below. The signal integrity can then be monitored for the specific PC board design and if required, the values can be adjusted.

5.5.9.1 Q_RCMP_SETUP_CONTROL

The QDR RCOMP Setup and Control Register.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED													SLEW_INDEX_SELECT	INC_DEC_INVERT	AR SOS	DR SOS	KCR SOS	DQDR SOS	RCOMP_LOCK	SLEW_RATE_TABLES	DIGITAL_FILTER_SELECT	RCOMPPRD		BLOCK_RCOMP_UPDATES	RCOMP_SM_DISABLE	FRCOMP	ROE_NMOS	ROE_PMOS

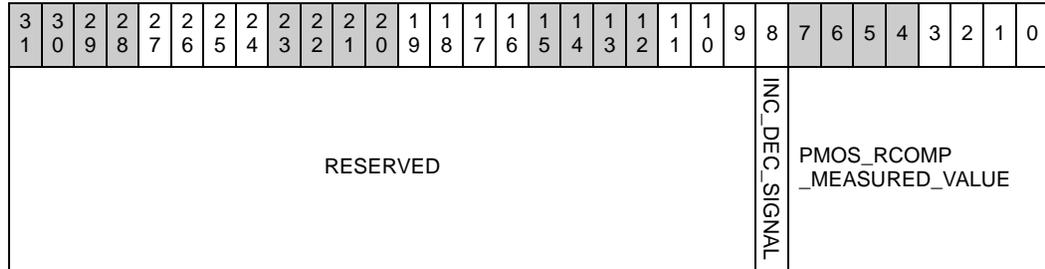
Bits	Field	Description	RW	Reset
[31:18]	RESERVED	Reserved	RO	0x0
[17]	SLEW_INDEX_SELECT	Use to select either RComp[6:3] or [5:2] to be the 4-bit index to the slew rate LUT. 0 = RCOMP[6:3] (default); 1 = RCOMP[5:2].	RW	0x0
[16]	INC_DEC_INVERT	Inc/Dec Invert: Invert the inc/dec signal from the pads.	RW	0x0

Bits	Field	Description	RW	Reset
[15]	ARSOS	Address RComp & SComp Override Select: When set will allow PMOS & NMOS RComp and SComp override register settings to over-ride the conditioned RComp values and slew rate LUT SComp values at the Address DRUPT output	RW	0x0
[14]	DRSOS	Data RComp & SComp Override Select: When set will allow PMOS & NMOS RComp and SComp override register settings to over-ride the conditioned RComp values and slew rate LUT SComp values at the Data DRUPT output.	RW	0x0
[13]	KCRSOS	K Clock RComp & SComp Override Select: When set will allow PMOS & NMOS RComp and SComp override register settings to over-ride the conditioned RComp values and slew rate LUT SComp values at the K clock DRUPT output.	RW	0x0
[12]	DQDRSOS	DQ Data RComp & SComp Override Select: When set will allow PMOS & NMOS RComp and SComp override register settings to over-ride the conditioned RComp values and slew rate LUT SComp values at the DQ Data DRUPT output.	RW	0x0
[11]	RCOMP_LOCK	RCOMP Lock Bit: Once this bit is set, any further reads or writes to registers with names beginning Q_RCMP will be ignored.	RW	0x0
[10]	SLEW_RATE_TABLES	Slew Rate Tables Programmed: This field provides a simple mechanism for SW and HW to know that the Slew Rate tables are programmed to the intended values. This field should be set by BIOS. It is currently ignored by hardware.	RW	0x0
[9:8]	DIGITAL_FILTER_SELECT	Digital Filter Select: Selects the amount of digital filtering to be applied to each SM RCOMP measurement cycle. This RCOMP state machine will not allow the 8-bit RCOMP values to change by more than the value programmed here. The default value of 00 selects an infinite clamp value, which effectively disables the filter. 00 — Infinite clamp value (default). Disable digital filter, the RComp pad tracks the buffer) 01 — Clamp value of 01h (Limited to one count of change at the RComp pad per cycle) 10— Clamp value of 02h (Limited to two counts of change at the RComp pad per cycle) 11 — Clamp value of 04h (Limited to four counts of change at the RComp pad per cycle)	RW	0x0

Bits	Field	Description	RW	Reset
[7:5]	RCOMPPRD	RCOMP Period (RCOMPPRD): This field controls the time period of the automatic hardware RCOMP operation being performed on the all enabled interface. The value programmed in this register will not take affect until the current RCOMP timer value expires. After each RCOMP time period, a new period will be loaded if this register has been written. Note: When the output is grounded (encoding = 111), the RCOMP period is 256 clocks. The values in the table are valid at 250MHz. Scaling should be applied for any other frequency. 000— Every 16ms 001 — Every 8ms 010 —Every 4ms 011 — Every 2 ms (default) 100 — Every 1ms 101 — Every 0.50 ms 110 — Every 0.25 ms 111 — Output is grounded.(Used with bit 2, below)	RW	0x3
[4]	BLOCK_RCOMP_UPDATES	Block RCOMP Updates: (Block at pads) 0 = Normal Operation (updates happen as needed). 1 = Blocks all updates to the DRUPT's and auto pad up-dates with DRrcomp_pad_update. DRrcomp_IDLE is still functional but the updated data is stale.	RW	0x0
[3]	RCOMP_SM_DISABLE	RComp State Machine Disable: (Block at WAIT) 0 — Normal Operation 1 — RComp cycle will stall at WAIT. All signals that enable the pad groups to update are at logic 0 and all RComp logic is frozen.	RW	0x0
[2]	FRCOMP	Force RCOMP Operation (FRCOMP): Used for Validation/DV. When a '1' is written to this bit a single RCOMP operation is performed on all enabled interfaces (Address, Data, K Clock, DQ inputs) simultaneously. Writing a '0' to this bit has no affect. This value is not stored. This register works in conjunction with RCOMP Period, bits 7:5 above.	RW	0x0
[1]	ROE_NMOS	RCOMP Override Enable for NMOS. Over ride occurs in the Eval block between the RComp counter and the digital filter input. The value stored in the NMOS RCOMP register is fed back to the NMOS RComp buffer and forward into the digital filter logic.	RW	0x0
[0]	ROE_PMOS	RCOMP Override Enable for PMOS. Over ride occurs in the Eval block between the RComp counter and the digital filter input. The value stored in the PMOS RCOMP register is fed back to the PMOS RComp buffer and forward into the digital filter logic.	RW	0x0

5.5.9.2 Q_RCMP_PMOS_MEASURED

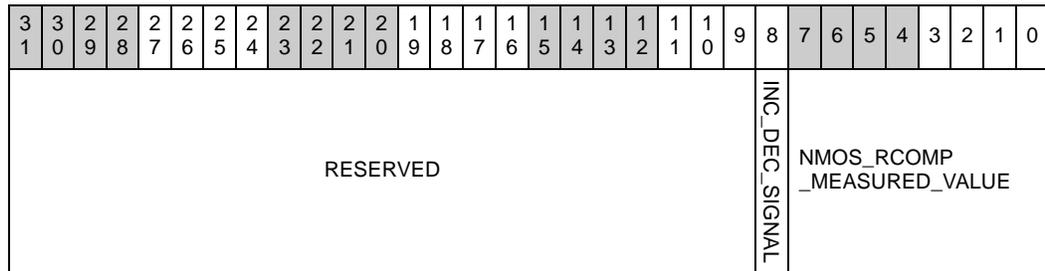
In normal operation, this register contains the current 8-bit RCOMP value for the PMOS drivers.



Bits	Field	Description	RW	IXP2400 Reset	IXP2800 Reset
[31:9]	RESERVED	Reserved	RO	0x0	0x0
[8]	INC_DEC_SIGNAL	Inc/dec signal in PMOS Eval block.	RO	1	dep
[7:0]	PMOS_RCOMP_MEASURED_VALUE	PMOS RCOMP Measured Value. A instrumented signal that is used to view the servo-mechanism.	RO	0x37	dep

5.5.9.3 Q_RCMP_NMOS_MEASURED

In normal operation, this register contains the current 8-bit RCOMP value for the NMOS drivers.



Bits	Field	Description	RW	IXP2400 Reset	IXP2800 Reset
[31:9]	RESERVED	Reserved	RO	0x0	0x0
[8]	INC_DEC_SIGNAL	Inc/dec signal in NMOS Eval block.	RO	0x37	dep
[7:0]	NMOS_RCOMP_MEASURED_VALUE	NMOS RCOMP Measured Value. A instrumented signal that is used to view the servo-mechanism.	RO	0x37	dep

5.5.9.4 Q_RCMP_PMOS_OVERRIDE

In normal operation, this register contains the current 8-bit RCOMP value for the PMOS drivers.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
ADDRESS_OVERRIDE								DATA_OVERRIDE								K_CLOCK_OVERRIDE				DQ_DATA_OVERRIDE										

Bits	Field	Description	RW	IXP2400 Reset	IXP2800 Reset
[31:24]	ADDRESS_OVERRIDE	Address PMOS RCOMP Override Value	RW	0x00	0x1C
[23:16]	DATA_OVERRIDE	Data PMOS RCOMP Override Value.	RW	0x00	0x1C
[15:8]	K_CLOCK_OVERRIDE	K Clock PMOS RCOMP Override Value.	RW	0x00	0x1C
[7:0]	DQ_DATA_OVERRIDE	DQ Data PMOS RCOMP Override Value.	RW	0x00	0x1C

5.5.9.5 Q_RCMP_NMOS_OVERRIDE

In normal operation, this register contains the 8-bit override values for the RComp NMOS drivers.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
ADDRESS_OVERRIDE								DATA_OVERRIDE								K_CLOCK_OVERRIDE				DQ_DATA_OVERRIDE										

Bits	Field	Description	RW	IXP2400 Reset	IXP2800 Reset
[31:24]	ADDRESS_OVERRIDE	Address NMOS RCOMP Override Value	RW	0x00	0x1C
[23:16]	DATA_OVERRIDE	Data NMOS RCOMP Override Value.	RW	0x00	0x1C
[15:8]	K_CLOCK_OVERRIDE	K Clock NMOS RCOMP Override Value.	RW	0x00	0x1C
[7:0]	DQ_DATA_OVERRIDE	DQ Data NMOS RCOMP Override Value.	RW	0x00	0x1C

5.5.9.6 Q_RCMP_PMOS_NMOS_SCOMP_OVERRIDE (IXP2400 and IXP2800 Rev A)

In normal operation, this register contains the 8-bit override values for the PMOS and NMOS SComp drivers.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	
ADDRESS_PMOS_SCOMP_OVERRIDE				ADDRESS_NMOS_SCOMP_OVERRIDE				DATA_PMOS_SCOMP_OVERRIDE				DATA_NMOS_SCOMP_OVERRIDE				K_CLOCK_PMOS_SCOMP_OVERRIDE				K_CLOCK_NMOS_SCOMP_OVERRIDE				DQ_DATA_PMOS_SCOMP_OVERRIDE				DQ_DATA_NMOS_SCOMP_OVERRIDE			

Bits	Field	Description	RW	IXP2400 Reset	IXP2800 Reset
[31:28]	ADDRESS_PMOS_SCOMP_OVERRIDE	Address PMOS SCOMP Override Value.	RW	0x0	0x03
[27:24]	ADDRESS_NMOS_SCOMP_OVERRIDE	Address NMOS SCOMP Override Value.	RW	0x0	0x03
[23:20]	DATA_PMOS_SCOMP_OVERRIDE	Data PMOS SCOMP Override Value.	RW	0x0	0x03
[19:16]	DATA_NMOS_SCOMP_OVERRIDE	Data NMOS SCOMP Override Value.	RW	0x0	0x03
[15:12]	K_CLOCK_PMOS_SCOMP_OVERRIDE	K Clock PMOS SCOMP Override Value.	RW	0x0	0x03
[11:8]	K_CLOCK_NMOS_SCOMP_OVERRIDE	K Clock NMOS SCOMP Override Value.	RW	0x0	0x03
[7:4]	DQ_DATA_PMOS_SCOMP_OVERRIDE	DQ Data PMOS SCOMP Override Value.	RW	0x0	0x03
[3:0]	DQ_DATA_NMOS_SCOMP_OVERRIDE	DQ Data NMOS SCOMP Override Value.	RW	0x0	0x03

5.5.9.7 Q_RCMP_PMOS_NMOS_SCOMP_OVERRIDE(IXP2800 Rev B)

In normal operation, this register contains the 8-bit override values for the PMOS and NMOS SComp drivers.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0		
ADDRESS_PMOS_SCOMP_OVERRIDE				ADDRESS_NMOS_SCOMP_OVERRIDE				DATA_PMOS_SCOMP_OVERRIDE				DATA_NMOS_SCOMP_OVERRIDE				K_CLOCK_PMOS_SCOMP_OVERRIDE				K_CLOCK_NMOS_SCOMP_OVERRIDE				RESERVED	QDR_WR_PTR_RST_SEL_MODE	RESERVED		QDR_RX_DLL_DISABLE	QDR_TX_DLL_DISABLE	RESERVED	

Bits	Field	Description	RW	IXP2800 Reset
[31:28]	ADDRESS_PMOS_SCOMP_OVERRIDE	Address PMOS SCOMP Override Value.	RW	0x03
[27:24]	ADDRESS_NMOS_SCOMP_OVERRIDE	Address NMOS SCOMP Override Value.	RW	0x03
[23:20]	DATA_PMOS_SCOMP_OVERRIDE	Data PMOS SCOMP Override Value.	RW	0x03
[19:16]	DATA_NMOS_SCOMP_OVERRIDE	Data NMOS SCOMP Override Value.	RW	0x03
[15:12]	K_CLOCK_PMOS_SCOMP_OVERRIDE	K Clock PMOS SCOMP Override Value.	RW	0x03
[11:8]	K_CLOCK_NMOS_SCOMP_OVERRIDE	K Clock NMOS SCOMP Override Value.	RW	0x03
[7]	RESERVED	RESERVED	RW	0
[6]	QDR_WR_PTR_RST_SEL_MODE	Disables the phase comparator in the reset circuit for input FIFO write pointer 0 - Enable phase comparator 1 - Disable phase comparator	RW	0
[5:4]	RESERVED	RESERVED	RW	b11
[3]	QDR_RX_DLL_DISABLE	Disables the input master and slave DLL 1 - Disable 0 - Enable	RW	0
[2]	QDR_TX_DLL_DISABLE	Disables the internal 2X clock generator clk2g 1 - Disable 0 - Enable	RW	0
[1:0]	RESERVED	RESERVED	RW	b11

5.5.9.8 Q_RCMP_STRENGTH_SLEW_INDEX_SEL

This register controls the Slew Rate Index Clamp Enable and drive strength of the I/O buffers for the Address, Data, K Clock and DQ data signal groups. The four-bit strength setting is described in the table below.

Table 5-18. Strength Control Settings

Value	Setting	Value	Setting
0000	0.125	1000	1.25
0001	0.250	1001	1.50
0010	0.375	1010	2.00
0011	0.500	1011	2.125
0100	0.625	1100	2.25
0101	0.750	1101	2.50
0110	1.00	1110	3.00
0111	1.125	1111	4.00

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	
RESERVED												ADDR_SLEW_RATE_INDEX	ADDR_STRENGTH_CONTROL				DATA_SLEW_RATE_INDEX	DATA_STRENGTH_CONTROL				K_CLOCK_SLEW_RATE_INDEX	K_CLOCK_STRENGTH_CONTROL				DQ_DATA_SLEW_RATE_INDEX	DQ_DATA_STRENGTH_CONTROL			

Bits	Field	Description	RW	IXP2400 Reset	IXP2800 Reset
[31:20]	RESERVED	Reserved	RO	0x0	0x00
[19]	ADDR_SLEW_RATE_INDEX	Address Slew Rate Index RCOMP Clamp Enable. This bit causes the RCOMP[6:2] field that will be used to index into the Slew Rate tables to be clamped to "11111" whenever RCOMP[7] is set. If this bit is clear, then only RCOMP[6:2] will be used to index the Slew Rate tables regardless of the value on RCOMP[7].0: No RCOMP Clamping1: RCOMP Clamping Enabled.	RW	0x0	0x0
[18:15]	ADDR_STRENGTH_CONTROL	Address Strength Control.	RW	0x0	0x6
[14]	DATA_SLEW_RATE_INDEX	Data Slew Rate Index RCOMP Clamp Enable. This bit causes the RCOMP[6:2] field that will be used to index into the Slew Rate tables to be clamped to "11111" whenever RCOMP[7] is set. If this bit is clear, then only RCOMP[6:2] will be used to index the Slew Rate tables regardless of the value on RCOMP[7].0: No RCOMP Clamping1: RCOMP Clamping Enabled.	RW	0x0	0x0
[13:10]	DATA_STRENGTH_CONTROL	Data Strength Control.	RW	0x0	0x6
[9]	K_CLOCK_SLEW_RATE_INDEX	K Clock Slew Rate Index RCOMP Clamp Enable. This bit causes the RCOMP[6:2] field that will be used to index into the Slew Rate tables to be clamped to "11111" whenever RCOMP[7] is set. If this bit is clear, then only RCOMP[6:2] will be used to index the Slew Rate tables regardless of the value on RCOMP[7].0: No RCOMP Clamping1: RCOMP Clamping Enabled.	RW	0x0	0x0
[8:5]	K_CLOCK_STRENGTH_CONTROL	K Clock Strength Control.	RW	0x0	0x6
[4]	DQ_DATA_SLEW_RATE_INDEX	DQ Data Slew Rate Index RCOMP Clamp Enable. This bit causes the RCOMP[6:2] field that will be used to index into the Slew Rate tables to be clamped to "11111" whenever RCOMP[7] is set. If this bit is clear, then only RCOMP[6:2] will be used to index the Slew Rate tables regardless of the value on RCOMP[7].0: No RCOMP Clamping1: RCOMP Clamping Enabled.	RW	0x0	0x0
[3:0]	DQ_DATA_STRENGTH_CONTROL	DQ Data Strength Control.	RW	0x0	0x6

5.5.9.9 Q_RCMP_ADDR_PMOS_PU_OFFSET

This value is a signed offset applied to the final PMOS RCOMP/Strength value determined for the address signal group.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						ADDR_PMOS_OFFSET										

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0x00
[7:0]	ADDR_PMOS_OFFSET	Signed 8-bit offset to be applied to the PMOS drive strength for the Address signal group.	RW	0x00

5.5.9.10 Q_RCMP_ADDR_NMOS_PD_OFFSET

This value is a signed offset applied to the final NMOS RCOMP/Strength value determined for the Address signal group.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						ADDR_NMOS_OFFSET										

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0x00
[7:0]	ADDR_NMOS_OFFSET	Signed 8-bit offset to be applied to the NMOS drive strength for the Address signal group.	RW	0x00

5.5.9.11 Q_RCMP_DATA_PMOS_PU_OFFSET

This value is a signed offset applied to the final PMOS RCOMP/Strength value determined for the Data signal group.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						DATA_PMOS_OFFSET										

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0x00
[7:0]	DATA_PMOS_OFFSET	Signed 8-bit offset to be applied to the PMOS drive strength for the Data signal group.	RW	0x00

5.5.9.12 Q_RCMP_DATA_NMOS_PD_OFFSET

This value is a signed offset applied to the final NMOS RCOMP/Strength value determined for the Data signal group.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						DATA_NMOS_OFFSET										

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0x00
[7:0]	DATA_NMOS_OFFSET	Signed 8-bit offset to be applied to the NMOS drive strength for the Data signal group.	RW	0x00

5.5.9.13 Q_RCMP_K_CLK_PMOS_PU_OFFSET

This value is a signed offset applied to the final PMOS RCOMP/Strength value determined for the K Clock signal group.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						K_CLK_PMOS_OFFSET										

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0x00
[7:0]	K_CLK_PMOS_OFFSET	Signed 8-bit offset to be applied to the PMOS drive strength for the K Clock signal group.	RW	0x00

5.5.9.14 Q_RCMP_KCLK_NMOS_PD_OFFSET

This value is a signed offset applied to the final NMOS RCOMP/Strength value determined for the K Clock signal group.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						K_CLK_NMOS_OFFSET										

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0x00
[7:0]	K_CLK_NMOS_OFFSET	Signed 8-bit offset to be applied to the NMOS drive strength for the K Clock signal group.	RW	0x00

5.5.9.15 Q_RCMP_DQ_PMOS_PU_OFFSET

This value is a signed offset applied to the final PMOS RCOMP/Strength value determined for the DQ signal group.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																				DQ_PMOS_OFFSET										

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0x00
[7:0]	DQ_PMOS_OFFSET	Signed 8-bit offset to be applied to the PMOS drive strength for the DQ signal group.	RW	0x00

5.5.9.16 Q_RCMP_DQ_NMOS_PD_OFFSET

This value is a signed offset applied to the final NMOS RCOMP/Strength value determined for the DQ signal group.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																				DQ_NMOS_OFFSET										

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0x00
[7:0]	DQ_NMOS_OFFSET	Signed 8-bit offset to be applied to the NMOS drive strength for the DQ signal group.	RW	0x00

5.5.9.17 Q_RCMP_PMOS_NMOS_VERT_OVERRIDE

In normal operation, this register contains the 8-bit vertical override values for the RComp PMOS & NMOS drivers.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED														NMOS_RCOMP_OVERRIDE				PMOS_RCOMP_OVERRIDE									

Bits	Field	Description	RW	IXP2400 Reset	IXP2800 Reset
[31:16]	RESERVED	Reserved	RO	0x0	0x00
[15:8]	NMOS_RCOMP_OVERRIDE	NMOS RCOMP Override Value	RW	0x0	0x1C
[7:0]	PMOS_RCOMP_OVERRIDE	PMOS RCOMP Override Value	RW	0x0	0x1C

5.5.9.18 Slew Rate Tables

Each SRAM controller supports the following Slew Lookup Tables. Each table consists of four registers (designated by the # symbol in the name).

Table 5-19. SRAM Register Summary (where # = 0,1,2,3)

CSR name	Section
Q_RCMP_ADDR_PMOS_PU_SLEW_TABLE_#	These four registers contain the entire slew-rate lookup table for the pull-up (PMOS) devices in the I/O buffers for the Address signal group.
Q_RCMP_ADDR_NMOS_PD_SLEW_TABLE_#	These four registers contain the entire slew-rate lookup table for the pulldown (NMOS) devices in the I/O buffers for the Address signal group.
Q_RCMP_DATA_PMOS_PU_SLEW_TABLE_#	These four registers contain the entire slew-rate lookup table for the pull-up (PMOS) devices in the I/O buffers for the Data signal group.
Q_RCMP_DATA_NMOS_PD_SLEW_TABLE_#	These four registers contain the entire slew-rate lookup table for the pulldown (NMOS) devices in the I/O buffers for the Data signal group.
Q_RCMP_KCLK_PMOS_PU_SLEW_TABLE_#	These four registers contain the entire slew-rate lookup table for the pull-up (PMOS) devices in the I/O buffers for the K Clock signal group.
Q_RCMP_KCLK_NMOS_PD_SLEW_TABLE_#	These four registers contain the entire slew-rate lookup table for the pulldown (NMOS) devices in the I/O buffers for the K Clock signal group.
Q_RCMP_DQ_PMOS_PU_SLEW_TABLE_#	These four registers contain the entire slew-rate lookup table for the pull-up (PMOS) devices in the I/O buffers for the DQ signal group.
Q_RCMP_DQ_NMOS_PD_SLEW_TABLE_#	These four registers contain the entire slew-rate lookup table for the pulldown (NMOS) devices in the I/O buffers for the DQ signal group.

As shown in [Table 5-20](#) and [Table 5-21](#), the format of these registers are different for the IXP2400 and the IXP2800.

Table 5-20. Slew Table Format: IXP2400

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED															ADDR_xMOS_3			ADDR_xMOS_2			ADDR_xMOS_1			ADDR_xMOS_0							
RESERVED															ADDR_xMOS_7			ADDR_xMOS_6			ADDR_xMOS_5			ADDR_xMOS_4							
RESERVED															ADDR_xMOS_11			ADDR_xMOS_10			ADDR_xMOS_9			ADDR_xMOS_8							
RESERVED															ADDR_xMOS_15			ADDR_xMOS_14			ADDR_xMOS_13			ADDR_xMOS_12							

Note: The x in the name is equal to either P (for POS) or N (for NMOS) depending on the table.

Table 5-21. Slew Table Format: IXP2800

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
ADDR_xMOS_7			ADDR_xMOS_6			ADDR_xMOS_5			ADDR_xMOS_4			ADDR_xMOS_3			ADDR_xMOS_2			ADDR_xMOS_1			ADDR_xMOS_0										
ADDR_xMOS_15			ADDR_xMOS_14			ADDR_xMOS_13			ADDR_xMOS_12			ADDR_xMOS_11			ADDR_xMOS_10			ADDR_xMOS_9			ADDR_xMOS_8										
ADDR_xMOS_23			ADDR_xMOS_22			ADDR_xMOS_21			ADDR_xMOS_20			ADDR_xMOS_19			ADDR_xMOS_18			ADDR_xMOS_17			ADDR_xMOS_16										
ADDR_xMOS_31			ADDR_xMOS_30			ADDR_xMOS_29			ADDR_xMOS_28			ADDR_xMOS_27			ADDR_xMOS_26			ADDR_xMOS_25			ADDR_xMOS_24										

Note: The x in the name is equal to either P (for POS) or N (for NMOS) depending on the table.

The reset values for all the Slew Lookup Tables are undefined until they are programmed for the IXP2800. The recommended first time power-up values are defined in [Table 5-22](#).

Table 5-22. Slew Rate Table Recommended Initial Values (IXP2800)

Bits 31:16	Bits 15:12	Bits 11:8	Bits 7:4	Bits 3:0	Word Address Offset
0x3333	0x3	0x3	0x3	0x3	00h
0x3333	0x3	0x3	0x3	0x3	01h
0x3333	0x3	0x3	0x3	0x3	10h
0x3333	0x3	0x3	0x3	0x3	11h

The reset values for all the Slew Lookup Tables are undefined for the IXP2400. The recommended first time power-up values are defined in [Table 5-23](#).

Table 5-23. Slew Rate Table Recommended Initial Values (IXP2400)

Bits 31:16	Bits 15:12	Bits 11:8	Bits 7:4	Bits 3:0	Word Address Offset
0x0	0xc	0xc	0xc	0xc	00h
0x0	0xc	0xc	0xc	0xc	01h
0x0	0xc	0xc	0xc	0xc	10h
0x0	0xc	0xc	0xc	0xc	11h

5.5.10 QDR unit initialization

The recommended initial setup procedure differs between IXP2800 and IXP2400.

5.5.10.1 IXP2800 A Steppings QDR initial setup procedure

The section describes the control status register (CSR) initialization sequence to configure each SRAM controller for operation.

The IXP2800 Network processor integrates four QDRII compliant SRAM controllers. The following procedure explains how to program each SRAM controllers CSR's in order to initialize each channel for operation. Note that while values for each CSR are recommended within this procedure, the timing characteristics of each implementation will vary and hence the programmed values for each implementation may also vary. This procedure shows an example of the initialization sequence for Channel 0 this sequence must be repeated for all SRAM channels. The following procedure can be used to empirically determine the appropriate values.

1. Program the CLOCK_CONTROL register at address C0004A14 with the appropriate SRAM_CHN_CLK_RATIO value. For A0/A1 steppings, assuming the setting value is 0x033A_6666, CLOCK_CONTROL needs to be set to 0x0300_0000 first, then set it to 0x033A_6666; for A2 and future stepping the initial write of 0x0300_0000 is not required.
2. Each channel must then have its reset removed by clearing the respective reset register in the IXP_RESET_0 register at C0004A0C; bits 3 - 6 control channels 0 - 4 respectively.
3. Set QDR_DFT_CTRL register at address CC010200 with a value of 0x0000_2000.
4. Delay for 10ms prior to proceeding to step 5.
5. Set QDR_DFT_CTRL register at address CC010200 with a value of 0x0000_0000.
6. Setup the dynamic RCOMP logic by writing the Q_RCMP_SETUP_CONTROL register at address CC010300 with a value of 0x00000160. This setting will disable all bypass operations, set the digital filter to a single RComp change per RComp cycle and set the RComp Period to 2ms. These filter and period setting allow for 500 RComp changes per second.
7. Delay 10 ms prior to proceeding to step 8, this allows the RCOMP operations to complete.
8. Program the Q_RCMP_STRENGTH_SLEW_INDEX_SEL register at address CC010318 with a value of 0x000B5AD6.
9. Delay 10 ms prior to proceeding to step 10, this allows the RCOMP operations to complete.

10. The CSRs mentioned in this step are reset to a value of 0x00000000. This is already the recommended value for operation, but they are mentioned for completeness:

Q_RCMP_ADDR_PMOS_PU_OFFSET	Q_RCMP_K_CLK_PMOS_PU_OFFSET
Q_RCMP_ADDR_NMOS_PD_OFFSET	Q_RCMP_K_CLK_NMOS_PD_OFFSET
Q_RCMP_DATA_PMOS_PU_OFFSET	Q_RCMP_DQ_PMOS_PU_OFFSET
Q_RCMP_DATA_NMOS_PD_OFFSET	Q_RCMP_DQ_NMOS_PD_OFFSET

11. Initialize the slew rate tables to the value 0x33333333. (The # symbol in the register names can be 0 to 3 and indicate the four registers in the table.)

Q_RCMP_ADDR_PMOS_PU_SLEW_TABLE_#	Q_RCMP_KCLK_PMOS_PU_SLEW_TABLE_#
Q_RCMP_ADDR_NMOS_PD_SLEW_TABLE_#	Q_RCMP_KCLK_NMOS_PD_SLEW_TABLE_#
Q_RCMP_DATA_PMOS_PU_SLEW_TABLE_#	Q_RCMP_DQ_PMOS_PU_SLEW_TABLE_#
Q_RCMP_DATA_NMOS_PD_SLEW_TABLE_#	Q_RCMP_DQ_NMOS_PD_SLEW_TABLE_#

12. Delay 10 ms prior to proceeding to step 13, this allows the RCOMP operations to complete.
13. Complete the setup the dynamic RCOMP logic by writing the Q_RCMP_SETUP_CONTROL register at address CC010300 with a value of 0x0000D60.
14. Program the SRAM_CONTROL_00 Register at CC010000 with 0x00000100. This configures the channel for a size of 4MB and parity disabled.
15. Program the QDR_INTERNAL_PIPELINE_00 register at address CC010108 with a value of 0x00000002, which adds or removes one cycle of read latency. Please note that a value of two has been found empirically to be the best for 200 MHz operation; however this may vary per implementation. This value should be set to a value of 0x00000001 for 100/133MHz/QDRI operation. Again this may vary per implementation.
16. Program the SRAM_RX_DESKEW_00 register at address CC010244 with 0x00000001. Wait 10ms, then program SRAM_RX_DESKEW_00 register with 0x00000002. Wait 10ms. Continue incrementing the value after 10ms waits until reaching a final value of 0x00000008. This sets the number of delay elements in the read clock DLL to <value> + 1. It is recommended to only increment or decrement <value> in order to prevent the DLL from having to deal with a large step response where it is at risk of locking to a harmonic.
17. Program the SRAM_RX_DLL_00 register at address CC010228 with a value of 0x00000008, this adjusts the read clock DLL such that capture clock is centered in the middle of the receive data eye window. The optimal value for this register will depend primarily on two features of the board-level implementation. The first important consideration is the relationship of Q to CIN. It is typically desired to have the read capture strobe placed in the middle of the data valid window. Setting SRAM_RX_DLL_00 equal to SRAM_RX_DESKEW_00 achieves this if Q and CIN arrive simultaneously. At 200MHz, changing the SRAM_RX_DLL_00 value by one corresponds to moving the read capture strobe by ~65ps. So if a theoretical implementation had Q data lagging CIN by ~200ps, an SRAM_RX_DLL_00 value set to SRAM_RX_DESKEW_00 value + 3 = 0x0000000B would be optimal. The second relevant consideration is the C-CIN loop length. There is clock-crossing between the read capture strobe domain and the core's PLL derived clock domain. If the read strobe is pushed out too far (e.g. long CIN loop length or too high a value for SRAM_RX_DLL_00), the data capture occurs beyond a core clock boundary. To recover the correct data, the strobe may be left as is, and the pipeline delay increased (an increase in read latency), or the read capture strobe may be pulled earlier than the center of the data valid window.
18. Delay 10 ms prior to proceeding to step 19, this allows the Deskew circuit to lock prior to performing RCOMP operations.

19. The final step is to reset the RD_WR_PTR logic by programming the SRAM_RD_PTR_OFFSET_00 register at address CC010240 with a value of 0x00000000. Note for QDRI mode this register should be programmed to a value of 0x00000004.

At this point the channel should be configured correctly and its timing characteristics adjusted to compensate for latency incurred by the physical implementation. If data cannot be read and written correctly then it may be required to perform another/multiple iterations of this procedure to determine the appropriate values for QDR_INTERNAL_PIPELINE_00 and SRAM_RX_DLL_00 registers as described in step 17. It is also recommended that the range of operational values for SRAM_RX_DLL_00 register be determined empirically by stepping the programmed value in step 15. Note that once the channel has been initialized the SRAM_RX_DLL_00 register value can be modified without re-initializing the channel; however after any change to this register step 17 must also be performed.

5.5.10.2 IXP2800 B Steppings - QDR initial setup procedure

This procedure is an example of the initialization sequence for Channel 0 for *B* steppings. You must repeat this sequence for *all* QDR SRAM channels. Use the following procedure to empirically determine the appropriate values:

1. Program the CLOCK_CONTROL register at address C0004A14 with the appropriate SRAM_CHN_CLK_RATIO value.
2. Remove the reset from each channel by clearing the respective reset register in the IXP_RESET_0 register at C0004A0C; bits 3 – 6 control channels 0 – 3, respectively.
3. Enable the TX and RX DLLs. For the B stepping, individual enables are provided for the TX and RX DLLs in the QDR unit. The lower eight bits of Q_RCOM_PMOS_NMOS_SCOMP_OVERRIDE at offset 0x0314 have been modified to accommodate this feature. The previous functionality of this register and the registers DQ_DATA_PMOS_SCOMP_OVERRIDE and DQ_DATA_NMOS_SCOMP_OVERRIDE, have been hardwired to their reset value of 0x3. Use the following steps to enable the DLLs:

- a. Write a value of 0x1 to Q_RCOM_PMOS_NMOS_SCOMP_OVERRIDE[QDR_TX_DLL_DISABLE] to disable the TX DLL.
- b. Write a value of 0x1 to Q_RCOM_PMOS_NMOS_SCOMP_OVERRIDE[QDR_RX_DLL_DISABLE] to disable the RX DLL.
- c. Write a value of 0x00 to Q_RCOM_PMOS_NMOS_SCOMP_OVERRIDE[QDR_K_CLOCK_ENABLE] to disable the K/K# clocks.
- d. Delay 10 ms.
- e. Write a value of 0x0 to Q_RCOM_PMOS_NMOS_SCOMP_OVERRIDE[QDR_TX_DLL_DISABLE] to re-enable the TX DLL.
- f. Delay 10 ms.
- g. Write a value of 0x0 to Q_RCOM_PMOS_NMOS_SCOMP_OVERRIDE[QDR_RX_DLL_DISABLE] to re-enable the RX DLL.
- h. Delay 10 ms.

4. Set up the dynamic RCOMP logic by writing the Q_RCMP_SETUP_CONTROL register at address CC010300 with a value of 0x00000160. This setting disables all bypass operations, sets the digital filter to a single RComp change per RComp cycle, and sets the RComp Period to 2 ms. These filter and period setting allow for 500 RComp changes per second.
5. Program the Q_RCMP_STRENGTH_SLEW_INDEX_SEL register at address CC010318 with a value of 0x000B5AD6.
6. The following CSRs are reset automatically to a value of 0x00000000 (the recommended value for operation); they are listed here for completeness:

Q_RCMP_ADDR_PMOS_PU_OFFSET	Q_RCMP_K_CLK_PMOS_PU_OFFSET
Q_RCMP_ADDR_NMOS_PD_OFFSET	Q_RCMP_K_CLK_NMOS_PD_OFFSET
Q_RCMP_DATA_PMOS_PU_OFFSET	Q_RCMP_DQ_PMOS_PU_OFFSET
Q_RCMP_DATA_NMOS_PD_OFFSET	Q_RCMP_DQ_NMOS_PD_OFFSET

7. Initialize the following slew rate tables to the value 0x33333333. (The # symbol in the register names can be 0 to 3 and indicate the four registers in the table.)

Q_RCMP_ADDR_PMOS_PU_SLEW_TABLE_#	Q_RCMP_KCLK_PMOS_PU_SLEW_TABLE_#
Q_RCMP_ADDR_NMOS_PD_SLEW_TABLE_#	Q_RCMP_KCLK_NMOS_PD_SLEW_TABLE_#
Q_RCMP_DATA_PMOS_PU_SLEW_TABLE_#	Q_RCMP_DQ_PMOS_PU_SLEW_TABLE_#
Q_RCMP_DATA_NMOS_PD_SLEW_TABLE_#	Q_RCMP_DQ_NMOS_PD_SLEW_TABLE_#

8. Write the Q_RCMP_SETUP_CONTROL register at address CC010300 with a value of 0x00000D60.
9. Set QDR_DFT_CTRL¹ register at address CC010200 with a value of 0x0000_2000. This will allow updates to occur when the RCOMP_LOCK is asserted.
10. Delay 10ms
11. Write Q_RCMP_SETUP_CONTROL register at address CC010300 with a value of 0x00000560. This clears the RCOMP_LOCK bit.
12. Set QDR_DFT_CTRL¹ register at address CC010200 with a value of 0x0000_0000.
13. Delay 1ms
14. Write a value of 0x3 to Q_RCMP_PMOS_NMOS_SCOMP_OVERRIDE[QDR_K_CLOCK_ENABLE] to enable the K/K# clocks.
15. Complete the setup of the dynamic RCOMP logic by writing Q_RCMP_SETUP_CONTROL register at address CC010300 with a value of 0x00000D60.
16. Program the SRAM_CONTROL_00 register at CC010000, with 0x00000100. This configures the channel for a size of 4 MB with parity disabled.
17. Program the QDR_INTERNAL_PIPELINE_00 register at address CC010108, with a value of 0x00000002, which adds or removes one cycle of read latency. *Note:* it has been determined empirically, that a value of 2 is best for a 200 MHz operation; however this may vary per implementation. For 100 – 133 MHz/QDRI operations, this value should be set to 0x00000001. Again, this may vary per implementation.
18. Program the SRAM_RX_DESKEW_00 register at address CC010244, with 0x00000001. Wait 10 ms, then program this register with 0x00000002. Wait another 10 ms. Continue incrementing the value after intervals of 10 ms, until you reach a final value of 0x00000008.

1. This is an Intel Reserved test register and it is not documented in the Programmers Reference Manual.

This sets the number of delay elements in the read clock DLL to $0x00000008 + 1$. It is recommended that the final value be incremented or decremented *only* to prevent the DLL from having to handle a large step response, which makes it at risk of locking to a harmonic.

19. Program the SRAM_RX_DLL_00 register at address CC010228, with a value of $0x00000008$; this adjusts the read clock DLL so that the capture clock is centered in the middle of the receive data eye window. The optimal value for this register depends primarily on two features of the board-level implementation: the relationship of Q to CIN and the C – CIN loop length, as described in the following substeps:
 - a. For the relationship of Q to CIN, it is recommended that the read-capture strobe be placed in the middle of the data valid window. Setting SRAM_RX_DLL_00 equal to SRAM_RX_DESKEW_00 achieves this if Q and CIN arrive simultaneously. At 200 MHz, changing the SRAM_RX_DLL_00 value by 1 corresponds to moving the read-capture strobe by approximately 65 ps. For example, with an implementation that has Q data lagging CIN by approximately 200 ps, an SRAM_RX_DLL_00 value set to $SRAM_RX_DESKEW_00 + 3$ (equaling $0x0000000B$) would be optimal.
 - b. For the C – CIN loop length, there is clock-crossing between the read-capture strobe domain and the core's PLL-derived clock domain. If the read strobe is pushed out too far (for example, by a long CIN loop length or an excessively high value for SRAM_RX_DLL_00), the data capture occurs past a core clock boundary. To recover the correct data, the strobe may be left as is, and the pipeline delay increased (increasing read latency), or the read-capture strobe may be pulled earlier than the center of the data valid window.
20. Delay 10 ms prior to proceeding to 21., to allow the Deskew circuit to lock prior to performing RCOMP operations.
21. Reset the RD_WR_PTR logic by programming the SRAM_RD_PTR_OFFSET_00 register at address CC010240 with a value of $0x00000000$. *Note:* for QDRI mode this register should be programmed to a value of $0x00000004$.

For *B steppings*, the preceding sequence of steps should configure the channel with timing characteristics adjusted to compensate for latency incurred by the physical implementation. If data cannot be read and written correctly, then it may be necessary to perform one or more iterations of this procedure to determine the appropriate values for the SRAM_RX_DESKEW_00 and SRAM_RX_DLL_00 registers as described in 19.

It is recommended that the range of operational values for the SRAM_RX_DLL_00 register be determined empirically by stepping the programmed value in 19.

After the channel has been initialized, you can modify the SRAM_RX_DLL_00 register value without re-initializing the channel; however, after any change to this register, you must perform 20. and 21. again.

5.5.10.3 IXP2400 QDR initial setup procedure

For IXP2400, the recommended initial setup procedure is as follows:

1. Set the following slew rate tables to $0x0000cccc$. (The # symbol in the register names can be 0 to 3 and indicate the four registers in the table.)

Q_RCMP_ADDR_PMOS_PU_SLEW_TABLE_#	Q_RCMP_KCLK_PMOS_PU_SLEW_TABLE_#
Q_RCMP_ADDR_NMOS_PD_SLEW_TABLE_#	Q_RCMP_KCLK_NMOS_PD_SLEW_TABLE_#
Q_RCMP_DATA_PMOS_PU_SLEW_TABLE_#	Q_RCMP_DQ_PMOS_PU_SLEW_TABLE_#
Q_RCMP_DATA_NMOS_PD_SLEW_TABLE_#	Q_RCMP_DQ_NMOS_PD_SLEW_TABLE_#

2. Set the Q_RCMP_SETUP_CONTROL registers (offset 0x0300) to the value 0x00010060.
3. Set the Q_RCMP_STRENGTH_SLEW_INDEX_SEL registers (offset 0x0318) to the value 0x000318c6.
4. Set the QDR_RX_DLL registers (offset 0x0228) to the value 0x0000004c.
5. Set the QDR_RX_DESKEW registers (offset 0x0244) to the value 0x00000012.
6. Configure the SRAM_CONTROL registers (offset 0x0000).
7. Configure the QDR_RD_PTR_OFFSET registers (offset 0x0240). This configuration selects between QDR I and QDR II configurations.
8. Read the QDR_RD_PTR_OFFSET registers (offset 0x0240).

5.6 CSR Access Proxy (CAP)

The CAP address space includes CAP CSRs (Scratch, Hash, Fast Write, Global), Timers, UART, PMU, SlowPort CSRs, ME's Local CSRs, and the Reflector.

5.6.1 Scratchpad Memory CSRs (CAP CSR)

Table 5-24 shows the offset addresses of the Scratchpad Memory CSRs. Refer to [Chapter 4, "Address Maps"](#) for the base address and details on how they are accessed. These CSRs can be accessed by the Intel XScale® core, PCI and the MEs.

Each of the Rings has its own copy of the registers; the “#” symbol is the number of the ring. Substitute 0 through 0xF for the “#” to get the individual register names and addresses.

Note: There is an additional CSR associated with the Scratchpad memory which is located in the Intel XScale® core. Local CSRs and it provides the Ring Full status signals to the Intel XScale® Processor.

Warning: Scratch get and put operations must not be performed to ANY of the scratch rings at the same time ANY of the scratch ring configuration registers are being written otherwise the result of the get and put operations are unpredictable. A typical initialization sequence is one in which the Intel XScale® core initialize the rings once at system initialization time and then indicates to the Microengine that get and put operations can be performed.

Note: For IXP2800 Rev A only -- Before using the Scratchpad Memory, write a ‘1’ to bit 10 of register at address 0xfc, and then write a ‘0’ to bit 10 of the same register. This register is only used for manufacturing test and must be initialized prior to use. Since there are other test bits in this register, in order to write bit 10, the programmer should read the value of the register, modify bit 10, then write back the modified value of the register.

Table 5-24. Scratchpad Memory Register Summary

CSR name	Address	Description	Section
SCRATCH_RING_BASE_#	0x0#0	Base address of the Ring.	Section 5.6.1.1

Table 5-24. Scratchpad Memory Register Summary (Continued)

CSR name	Address	Description	Section
SCRATCH_RING_HEAD_#	0x0#4	Offset of head entry from Base.	Section 5.6.1.2
SCRATCH_RING_TAIL_#	0x0#8	Offset of tail entry from Base.	Section 5.6.1.3
RESERVED	0xFC		

5.6.1.1 SCRATCH_RING_BASE_# (# = 0 -15)

This register contains the Base Address in Scratchpad RAM for the Ring. This register must be written before the Ring can be used.

Warning: The Scratch CSRs must be initialized prior to performing Put and Get operations to any of the Scratch Rings otherwise the scratch operations are unpredictable.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
SIZE		RESERVED		RING_STATUS_FLAG	RESERVED										BASE				RESERVED											

Bits	Field	Description	RW	Reset
[31:30]	SIZE	Indicates the size of the Ring in 32-bit words. 00 = 128 01 = 256 10 = 512 11 = 1024	RW	Undef
[29:27]	RESERVED	Reserved	RO	0
[26]	RING_STATUS_FLAG	Rev A -- Reserved Rev B -- Indicates use of the Ring Status Flag 0 - Full. Ring Status Flag indicates that the Ring is above High Water Mark (which is based on the Ring Size field of this register). 1 - Empty. Ring Status Flag indicates that the Ring has no data on it.	RW	Undef
[25:14]	RESERVED	Reserved	RO	0
[13:9]	BASE	Ring Base Address. This value must be written to be consistent with the requirement that the Ring is aligned to its size. For example, if the Ring has 512 32-bit words, bits 10:9 must be written to 0.	RW	Undef
[8:0]	RESERVED	Reserved	RO	0

5.6.1.2 SCRATCH_RING_HEAD_# (# = 0 - 15)

This register contains the offset from the Ring Base of the current head. This is the next address to be read on a get. This register must be written to 0 before the Ring can be used. When the Ring is in use, the value is maintained by hardware, and can be read for debug and test.

Warning: The Scratch CSRs must be initialized prior to performing Put and Get operations to any of the Scratch Rings otherwise the scratch operations are unpredictable.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																	OFFSET										RESERVED			

Bits	Field	Description	RW	Reset
[31:12]	RESERVED	Reserved	RO	0
[11:2]	OFFSET	Offset: Note that this field is large enough for the largest size Ring. When the Ring is configured for a smaller size, only the bits of Offset as specified in Table 5-25 will be valid. If software uses this field, for example to calculate how many entries of the Ring are occupied, it must mask off unused bits.	RW	Undef
[1:0]	RESERVED	Reserved	RO	0

Table 5-25. Head/Tail Use and Full Threshold by Ring Size

Ring Size (# of 32-bit words)	Base Address	Head/Tail Offset	Full Threshold (Number of Empty Entries)
128	13:9	8:2	32
256	13:10	9:2	64
512	13:11	10:2	128
1024	13:12	11:2	256
NOTE: Note that bits [1:0] of the address are assumed to be 00.			

5.6.1.3 SCRATCH_RING_TAIL_# (#= 0 - 15)

This register contains the offset from the Ring Base of the current tail. This is the next address to be write on a put. This register must be written to 0 before the Ring can be used. When the Ring is in use, the value is maintained by hardware, and can be read for debug and test.

Warning: The Scratch CSRs must be initialized prior to performing Put and Get operations to any of the Scratch Rings otherwise the scratch operations are unpredictable.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																OFFSET											RESERVED				

Bits	Field	Description	RW	Reset
[31:12]	RESERVED	Reserved	RO	0
[11:2]	OFFSET	Offset: Note that this field is large enough for the largest size Ring. When the Ring is configured for a smaller size, only the bits of Offset as specified in Table 5-25 will be valid. If software uses this field, for example to calculate how many entries of the Ring are occupied, it must mask off unused bits.	RW	Undef
[1:0]	RESERVED	Reserved	RO	0

5.6.2 Hash Configuration (CAP CSR)

Table 5-26 shows the offset addresses of the Hash Multiplier CSRs. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can be accessed by the Intel XScale® core, PCI and the MEs.

The user should ensure that no hash operations are being performed when changing the multiplier value since there is no order guaranteed between writes to Hash_Multiplier and Hash operations. The Intel XScale® core perform Hash operations using the HASH_OP_x_x and HASH_DONE registers described in the Intel XScale® core Local CSR section.

Table 5-26. Hash Multiplier Register Summary

CSR name	Address	Description	Section
HASH_MULTIPLIER_48_0	0x00	Least significant 32 bits of 48-bit Hash Multiplier.	Section 5.6.2.1
HASH_MULTIPLIER_48_1	0x04	Most significant 16 bits of 48-bit Hash Multiplier.	

Table 5-26. Hash Multiplier Register Summary (Continued)

CSR name	Address	Description	Section
HASH_MULTIPLIER_64_0	0x08	Least significant 32 bits of 64-bit Hash Multiplier.	Section 5.6.2.2
HASH_MULTIPLIER_64_1	0x0C	Most significant 32 bits of 64-bit Hash Multiplier.	
HASH_MULTIPLIER_128_0	0x10	Least significant 32 bits of 128-bit Hash Multiplier.	Section 5.6.2.3
HASH_MULTIPLIER_128_1	0x14	Bits 32 to 63 of the 128-bit hash multiplier.	
HASH_MULTIPLIER_128_2	0x18	Bits 64 to 95 of the 128-bit hash multiplier.	
HASH_MULTIPLIER_128_3	0x1C	Most significant 32 bits of 128-bit Hash Multiplier.	

5.6.2.1 HASH_MULTIPLIER_48_# (# = 0,1)

These registers contain the programmable hash multiplier for generating 48-bit hash keys. HASH_MULTIPLIER_48_1 contains the most significant 16 bits of the 48-bit hash multiplier. HASH_MULTIPLIER_48_0 contains the least significant 32 bits of the 48-bit hash multiplier.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																HASH_MULT_48_1														

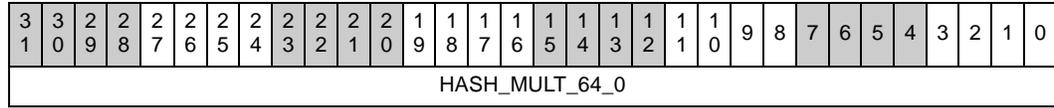
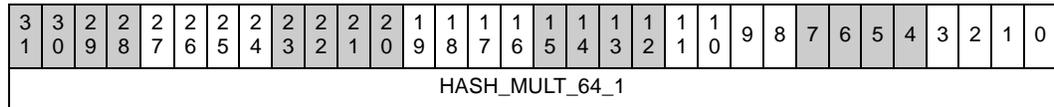
3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	
HASH_MULT_48_0																															

Bits	Field	Description	RW	Reset
[31:16]	RESERVED	Reserved	RO	0
[15:0]	HASH_MULT_48_1	The most significant 16 bits of the 48-bit hash multiplier.	RW	0

Bits	Field	Description	RW	Reset
[31:0]	HASH_MULT_48_0	The least significant 32 bits of the 48-bit hash multiplier.	RW	0

5.6.2.2 HASH_MULTIPLIER_64_# (# = 0,1)

These registers contain the programmable hash multiplier for generating 64-bit hash keys. HASH_MULTIPLIER_64_1 contains the most significant 32 bits of the 64-bit hash multiplier. HASH_MULTIPLIER_64_0 contains the least significant 32 bits of the 64-bit hash multiplier.



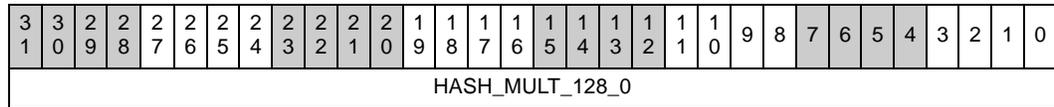
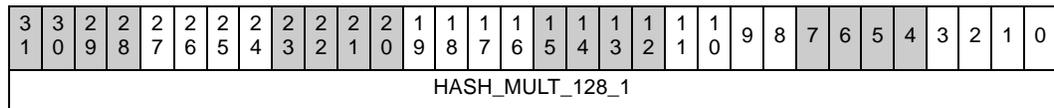
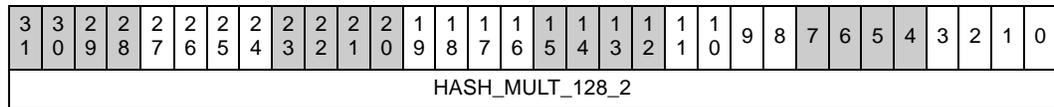
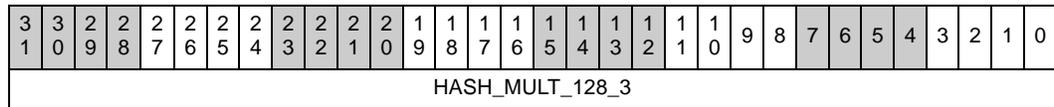
Bits	Field	Description	RW	Reset
[31:0]	HASH_MULT_64_1	The most significant 32 bits of the 64-bit hash multiplier.	RW	0

Bits	Field	Description	RW	Reset
[31:0]	HASH_MULT_64_0	The least significant 32 bits of the 64-bit hash multiplier.	RW	0

5.6.2.3 HASH_MULTIPLIER_128_# (# = 0,1,2,3)

These registers contain the programmable hash multiplier for generating 128-bit hash keys. HASH_MULTIPLIER_128_3 contains the most significant 32 bits of the 128-bit hash multiplier. HASH_MULTIPLIER_128_0 contains the least significant 32 bits of the 128-bit hash multiplier.

HASH_MULTIPLIER_128_1 and HASH_MULTIPLIER_128_2 contain bit 32 to bit 95 of the 128-bit hash multiplier.



Bits	Field	Description	RW	Reset
[31:0]	HASH_MULT_128_3	The most significant 32 bits (127 to 96) of the 128-bit hash multiplier.	RW	0

Bits	Field	Description	RW	Reset
[31:0]	HASH_MULT_128_2	Contains bits 64 to 95 of the 128-bit hash multiplier.	RW	0

Bits	Field	Description	RW	Reset
[31:0]	HASH_MULT_128_1	Contains bits 32 to 63 of the 128-bit hash multiplier.	RW	0

Bits	Field	Description	RW	Reset
[31:0]	HASH_MULT_128_0	The least significant 32 bits (31 to 0) of the 128-bit hash multiplier.	RW	0

5.6.3 Fast Write CSRs (CAP CSR)

Table 5-27 shows the offset addresses of the Inter-process Communication CSRs. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. Read and write operations can be performed on these CSRs by the Intel XScale® core, PCI and the MEs. Fast write operations can only be performed by the MEs.

Table 5-27. Inter-Process Communication Register Summary

CSR Name	Address	Description	Section
THD_MSG (Generic address)	0x000	Address for Microengine threads to write a message to their specific register. Refer to Note 1.	Section 5.6.3.1
THD_MSG_CLR_#_\$_& # = ME cluster number 0 to 1 \$ = ME number in cluster. 0 to 7 for IXP2800. 0 to 3 for IXP2400 & = thread number 0 to 7	0x100–0x2FC	Address to read and clear each individual THD_MSG. Refer to Note 1.	Section 5.6.3.2
THD_MSG_#_\$_& # = ME cluster number 0 to 1 \$ = ME number in cluster. 0 to 7 for IXP2800. 0 to 3 for IXP2400 & = thread number 0 to 7	0x500–0x6FC	Address to read each individual THD_MSG_#_\$_&. Refer to Note 1. For IXP2800, the offset for the 128 registers are $0x500 + (((cluster\# * 64) + (ME\# * 8) + Thread\#) * 4)$ For IXP2400, the offset for the 64 registers are $0x500 + (((cluster\# * 64) + (ME\# * 8) + Thread\#) * 4)$	Section 5.6.3.3
THD_MSG_SUMMARY_0_0	0x004 -	Bit vector registers that indicates which threads have new messages THD_MSG_SUMMARY_#_\$_ # = ME cluster number 0 to 1 \$ = register number 0 to 1 Refer to Note 1.	Section 5.6.3.4
THD_MSG_SUMMARY_0_1 (IXP2800 only)	0x008		
THD_MSG_SUMMARY_1_0	0x00C		
THD_MSG_SUMMARY_1_1 (IXP2800 only)	0x010		

Table 5-27. Inter-Process Communication Register Summary (Continued)

CSR Name	Address	Description	Section
SELF_DESTRUCT_0	0x014	Write bit number to set a bit in these registers and a read clears all the bits in the register	Section 5.6.3.5
SELF_DESTRUCT_1	0x018		
INTERTHREAD_SIG	0x01C	Writing a thread and signal number to this register generates a signal event to the thread	Section 5.6.3.6
XSCALE_INT_A	0xb20	Address for Microengine threads to set an interrupt to the Intel XScale® core.	Section 5.6.3.7
XSCALE_INT_B	0xb24		
<p>Note 1.</p> <p>Each Microengine thread can be programmed to write an 8-bit message to its own THD_MSG_#_\$_& register. The intent of these registers is to provide a mechanism to have the Microengine threads report their current processing status. The interpretation of the message is a software semantic between the sender and receiver.</p> <p>The numbering scheme of the Microengine threads involves the ME cluster, the ME number within the cluster and the thread number within each ME. There are two ME clusters for both IXP2800/2400. IXP2800 offers 8 MEs per cluster, with numbers 0 through 7. IXP2400 offers 4 MEs per cluster, with numbers 0 through 3.</p> <p>A Microengine thread writes this register using the fast_wr or cap[write] instruction with the generic THD_MSG register address. The data supplied with the instruction is written to the actual register associated with the Microengine thread number. CAP takes the generic address concatenates it with the ME and context number of the sender to create the specific address. The write will also set the bit corresponding to the sender in the THD_MSG_SUMMARY_#_\$_ Register.</p> <p>The cap[read] instruction or the Intel XScale® core read uses the actual THD_MSG register addresses to read these registers. There are two addresses to read THD_MSG. One will return the read data and clear the THD_MSG (and its corresponding THD_MSG_SUMMARY_#_\$_ bit), the other will return the read data and leave the contents of the register intact.</p> <p>The cap[read] instruction can use either the generic THD_MSG address or the actual thread specific THD_MSG_#_\$_& register addresses to read these registers. When the generic THD_MSG address is used for a read, CAP will determine the actual register in the same way as described above in the write description. There are two thread specific addresses for each THD_MSG; one will only read the data, the other will read the data and clear the THD_MSG register, and also clear the corresponding bit in the THD_MSG_SUMMARY_#_\$_ Register. Reading at the generic address does not do the clear function.</p>			

5.6.3.1 THD_MSG (Generic)

A ME thread writes this register using the cap[fast_wr] or cap[write] instruction to post a message. The data supplied with the instruction is written to the actual register associated with the Microengine thread number. CAP takes the generic address concatenates it with the ME and context number of the sender to create the specific address. The usage of this model to run common code on multiple threads. Without this address, SW would be required for figure out the specific register. The write will also set the bit corresponding to the sender in the THD_MSG_SUMMARY_#_\$_ Register. A thread can read it's own message register by reading this register. The read will result in a read to the associated THD_MSG_#_\$_& (by concatenating the ME and context number to create the specific address), however the usage of this model is that a thread typically does not read its own message register. The Intel XScale® core and PCI should not read this register since they do not have a message field reserved in these register.

Note that if the CTX or ME numbers are overridden by using an indirect_ref token in the cap command, the values of the CTX and the ME specified in the indirect reference are used in calculating the register address..I

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																				THD_MESSAGE									

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	THD_MESSAGE	Thread status	RW	0

5.6.3.2 THD_MSG_CLR_#_\$_& (# = {0,1}, \$ = {0,7 or 3}, & = {0,7})

= ME cluster number 0 to 1
 \$ = ME number in cluster. 0 to 7 for IXP2800 and 0 to 3 for IXP2400
 & = Thread number 0 to 7

Reading this register returns the thread status for the specified thread and clears the contents of this register as well as the corresponding summary bit in the THD_MSG_SUMMARY_#_\$_ register. Although this register can be written, the usage model is that this is read only register.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																				THD_MESSAGE									

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	THD_MESSAGE	Thread status	WRC	0

5.6.3.3 THD_MSG_#_\$_& (# = {0,1}, \$ = {0,7 or 3}, & = {0,7})

= ME cluster number 0 to 1
 \$ = ME number in cluster. 0 to 7 for IXP2800 and 0 to 3 for IXP2400
 & = Thread number 0 to 7

Reading this register returns the thread status for the specified thread. Although this register can be written, the usage model is that this is read only register.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																				THD_STATUS									

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
SELF_DEST_DATA																															

Bits	Field	Description	RW	Reset
[31:0]	SELF_DEST_DATA	Read/Write Data. A set bit corresponds to a value of 0 through 31 (decimal) written to this register during a <code>fast_wr</code> or <code>cap[write]</code> instruction. When the register is read, this field is cleared after reading.	WRC	0

5.6.3.6 INTERTHREAD_SIG

Any master can write a Microengine thread number to this register to signal an event to another thread. The value written to this register is broadcast to all Microengines. Each Microengine compares the Microengine number to its own number, and if it matches it sets the specified signal number for the specified thread.

Writes to INTERTHREAD_SIG are not queued at the Microengine. Multiple writes to the same thread ID may result in only 1 signal. Software must enforce a strict handshake between threads using this method.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																		ME CLUS	RESERVED	ME_NO	THD_NO	SIG									

Bits	Field	Description	RW	Reset
[31:12]	RESERVED	Reserved	W	0
[11]	ME CLUS	ME Cluster	W	0
[10]	RESERVED	Reserved when writing to this register this bit must always be 0	W	0
[9:7]	ME_NO	Microengine number that will be signaled. Valid IXP2800 ME numbers are 0 - 7. Valid IXP2400 ME numbers are 0 - 3.	W	0
[6:4]	THD_NO	Thread number (0-7) that will be signaled.	W	0
[3:0]	SIG	Number of the signal to deliver. 0x0 indicates no signal.	W	0

5.6.3.7 XSCALE_INT_# (# = A, B)

An ME writes any data to these registers to generate an Intel XScale® core interrupt. The ME and Context number are used to set a bit in the Intel XScale® core Local CSRs {IRQ,FIQ}THD_RAW_STATUS_\$_#, THD_RAW_STATUS_\$_ registers (refer to Section 5.10.1.18).



CAP takes the ME and context number of the sender to create the specific address. If the CTX or ME numbers are overridden by using an indirect_ref token in the cap command, the values of the CTX and the ME specified in the indirect reference are used in calculating the bit address.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
ANY DATA																															

Bits	Field	Description	RW	Reset
[31:0]	ANY_DATA	Any data can be written since it is ignored by CAP	WO	0

5.6.4 Global Control (CAP CSR)

Table 5-28 shows the offset addresses of the Global Control CSRs. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can be accessed by the Intel XScale® core, PCI and the MEs.

Table 5-28. Global Chassis Registers

Register Name	Address	Description	Section
PRODUCT_ID	0x00		Section 5.6.4.1
MISC_CONTROL	0x04		Section 5.6.4.2
MCCR (IXP2400 Only)	0x08		Section 5.6.4.3
IXP_RESET_0	0x0C		Section 5.6.4.4
IXP_RESET_1	0x10		Section 5.6.4.5
CLOCK_CONTROL	0x14		Section 5.6.4.6
STRAP_OPTIONS	0x18		Section 5.6.4.7
WATCHDOG_HISTORY	0x40		Section 5.6.4.8

5.6.4.1 PRODUCT_ID

PRODUCT_ID can be used by software to determine the type and revision level of the device.

The MAJOR REV is incremented for revisions that are visible to software, and may therefore require different versions of code. The MINOR REV is incremented for changes which are not visible to software, such as chip clock frequency improvements, bug fixes, etc. Chip documentation and Errata sheets will identify which version of the chip they pertain to by this register.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED											MAJ_PROD_TYPE				MIN_PROD_TYPE				MAJ_REV		MIN_REV										

Bits	Field	Description	RW	Reset
[31:21]	RESERVED	Reserved	RO	0
[20:16]	MAJ_PROD_TYPE	0 = IXP2000 others will be assigned as needed	RO	
[15:8]	MIN_PROD_TYPE	0 = IXP2800 2 = IXP2400 others will be assigned as needed	RO	
[7:4]	MAJ_REV	Current Revision. Starts at 0 for A stepping. B stepping is 1.	RO	
[3:0]	MIN_REV	0 = Rev A0/B0 1 = Rev A1 2 = Rev A2	RO	

5.6.4.2 MISC_CONTROL

This register contains control for miscellaneous functions.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED										PCI_CMD_PRIO				XSCALE_CMD_PRIO				RESERVED				FLASH_WR_EN	FLASH_ALIAS_DISABLE	TIMESTAMP_EN	CLK_DISABLE					

Bits	Field	Description	RW	Reset
[31:26]	RESERVED	Reserved	RO	0
[25:21]	PCI_CMD_PRIO	Priority for PCI in Command Bus Arbiter. The counter counts down one every cycle when PCI is requesting, and when it hits zero PCI has highest Command Bus priority. This field is the value that is loaded into the counter when it has counted down to zero. Lower values give PCI higher priority.	RW	0
[20:16]	XSCALE_CMD_PRIO	Priority for the Intel XScale® core in Command Bus Arbiter. The counter counts down one every cycle when the Intel XScale® core is requesting, and when it hits zero the Intel XScale® core has highest Command Bus priority. This field is the value that is loaded into the counter when it has counted down to zero. Lower values give the Intel XScale® core higher priority.	RW	0
[15:10]	RESERVED	Reserved	RO	0
9	FLASH_WR_EN	This bit provides some protection against software bugs accidentally causing a write to flash space which overwrites good data. 0—Writes to flash address space are discarded by the flash controller. 1—Writes to flash address are done as normal by the flash controller.	RW	0

Bits	Field	Description	RW	Reset
8	FLASH_ALIAS_DISABLE	This bit allows the Intel XScale® core to be able to boot from address 0, and to later put vectors into writable memory at address 0. 0—the flash ROM appears at address 0, as well as its normal address. 1—DRAM appears at address 0.	RW	0
7	TIMESTAMP_EN	This bit enables the Timestamp in the Microengines to count. Writing to this bit enables/disables all Timestamps at the same time, so they can be kept in sync. 0—Timestamps do not advance. 1—Timestamps advance.	RW	0
[6:0]	CLK_DISABLE	Control clocks to memory controller units. May be used to save power if a memory channel is not being used. Only valid for IXP2800. Reserved on IXP2400. When 0—Unit clock is enabled. When 1—Unit clock is disabled. Bit map: <ul style="list-style-type: none"> • 6 DRAM Channel 2 • 5 DRAM Channel 1 • 4 DRAM Channel 0 • 3 SRAM Channel 3 • 2 SRAM Channel 2 • 1 SRAM Channel 1 • 0 SRAM Channel 0 	RW	0

5.6.4.3 MSF Clock Control CSR (MCCR) - IXP2400 only

MSF Clock Control selects the clock ratio for the four IXP2400 MSF RX0, RX1, TX0, and TX1 PLL. This register must be programmed before accesses to the MSF.

These steps must be followed:

During power on initialization

1. PLL in PLL disable mode (MCCR[MSF_POWERDOWN] =1 default)
2. PLL in bypass mode (MCCR[MSF_BYPASS_SEL] =1 default)
3. Software Set up each of MSF clock ratio (MCCR[MSF_CLKCFG])
4. Disable the PLL bypass mode (MCCR[MSF_BY_PASS_SEL] = 0)
5. Turn on PLL(MCCR[MSF_POWERDOWN] =0)
6. Wait for MSF PLL lock (MCCR[MSF_PLL_LOCK] = 1)
7. Enable MSF block (via MSF_Rx_Control[Rx_En] or MSF_Tx_Control[Tx_En])
8. Initialize the MSF CSR registers



Bits	Field	Description	RW	Reset
[31:28]	MSF_PLL_LOCK	<p>This field is read only and is intended to provide visibility of the PLL LOCK output signal from the MSF PLLs. 0 = PLL is not locked 1 = PLL is locked</p> <p>[31] - TX PLL 1 (channels 2/3) [30] - TX PLL 0 (channels 0/1) [29] - RX PLL 1 (channels 2/3) [28] - RX PLL 0 (channels 0/1)</p>	RO	0x0
[27:26]		Reserved	RO	0x0
[25]	MSF_TX_CLK_MODE	<p>Used to configure the number of clocks on the transmit interface.</p> <p>0 - single clock mode; only TXCLK01 clock input is used; TXCLK23 clock input should be tied low</p> <p>1- dual clock mode; both TXCLK01 and TXCLK23 clock inputs are used</p>	RW	0x0
[24]	MSF_RX_CLK_MODE	<p>Used to configure the number of clocks on the receive interface.</p> <p>0 - single clock mode; only RXCLK01 clock input is used; RXCLK23 clock input should be tied low</p> <p>1- dual clock mode; both RXCLK01 and RXCLK23 clock inputs are used</p>	RW	0x0

Bits	Field	Description	RW	Reset
[23:16]	MSF_CLKCFG	<p>This field is used to set the multiplier/divisor for the PLLs. The PLL operates at the external clock multiplied by the value of the selected multiplier. The output of the PLL is then divided by the same value to recreate the original clock frequency.</p> <p>00: 48 (nominally 25 MHz) 01: 24 (nominally 50 MHz) 10: 16 (nominally 104 MHz) 11: 12 (nominally 125 MHz)</p> <p>For clock frequencies other than the nominal frequencies, select the setting for the closest value. For example, for 80 MHz, use "10" for a multiplier/divisor of 16. The requirement is that the clock frequency times the selected multiplier should fall in the range of 1 to 2 GHz to achieve lock.</p> <p>[23:22] - TX PLL 1 (channels 2/3) [21:20] - TX PLL 0 (channels 0/1) [19:18] - RX PLL 1 (channels 2/3) [17:16] - RX PLL 0 (channels 0/1)</p> <p>These steps must be followed: During power on initialization 1. PLL in PLL disable mode (MCCR[MSF_POWERDOWN] =1) 2. PLL in bypass mode (MCCR[MSF_BYPASS_SEL] =1) 3. Software Set up each of MSF clock ratio (MCCR[MSF_CLKCFG] bits) 4. Disable the PLL bypass mode (MCCR[MSF_BY_PASS_SEL] = 0) 5. Turn on PLL(MCCR[MSF_POWERDOWN] =0) 6. Wait for MSF PLL lock (MCCR[MSF_PLL_LOCK] = 1) 7. Initialize the MSF 8. Enable MSF block (via MSF_Rx_Control[Rx_En] or MSF_Tx_Control[Tx_En])</p>	RW	0x0
[15:12]	MSF_BYPASS_SEL	<p>Select MSF PLL bypass It should be in bypass mode during power on initialization.</p> <p>0: no bypass (Select PLL clock output) 1: bypass (Select external reference clock)</p> <p>[15] - TX PLL 1 (channels 2/3) [14] - TX PLL 0 (channels 0/1) [13] - RX PLL 1 (channels 2/3) [12] - RX PLL 0 (channels 0/1)</p>	RW	0xF



Bits	Field	Description	RW	Reset
[11:8]	MSF_DIV_RESET	<p>This is used to reset the divider and should be used only in test mode. To reset the divider, software must first write a 1, then a 0.</p> <p>0: reset not asserted 1: reset asserted</p> <p>[11] - TX PLL 1 (channels 2/3) [10] - TX PLL 0 (channels 0/1) [9] - RX PLL 1 (channels 2/3) [8] - RX PLL 0 (channels 0/1)</p>	RW	0x0
[7:4]	MSF_POWERDOWN (IDDQ Enable)	<p>MSF_POWERDOWN</p> <p>This is used for MSF PLL disable during the power on initialization. This is also can be used during IDDQ/leakage tests.</p> <p>0: no power down 1: power down</p> <p>[7] - TX PLL 1 (channels 2/3) [6] - TX PLL 0 (channels 0/1) [5] - RX PLL 1 (channels 2/3) [4] - RX PLL 0 (channels 0/1)</p>	RW	0xF
[3:0]	MSF_ICCTEST	<p>This is used to power down the differential amps in the clock buffer and is intended to be used only in IDDQ/leakage tests.</p> <p>0 = normal operation 1 = power down</p> <p>[3] - TX PLL 1 (channels 2/3) [2] - TX PLL 0 (channels 0/1) [1] - RX PLL 1 (channels 2/3) [0] - RX PLL 0 (channels 0/1)</p>	RW	0x0

5.6.4.4 IXP_RESET_0

Software reset control consists of two 32-bit registers: IXP_RESET_0 and IXP_RESET_1. IXP_RESET_0 is used to reset everything except Microengines. IXP_RESET_1 is used to reset Microengines. Software initiated system resets should be performed by setting the IXP_RESET_0[15] (RST_ALL). This is equivalent to a hard reset. Some reset bits in this register are for Intel testing only and should never be set. Other reset bits can be used to reset the unit during run time (if the unit is in a quiescent state) or hold units in reset if the units are not used. These registers are Read/Write by both PCI host and the Intel XScale® processor.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0				
RESERVED																RESERVED	MSF	SRAM_3	SRAM_2	SRAM_1	SRAM_0	PCI_RST	PCI	XSCALE									
																WATCHDOG	CRYPTO_UNIT	EXT_RESET_EN	INIT_COMP	CMD_ARB	SBUS_ARB	DBUS_ARB	SHaC	RST_ALL	EXT_RST	DRAM_3	DRAM_2	DRAM_1	DRAM_0				

Bits	Field	Description	RW	IXP 28x0 Reset	IXP 2400 Reset
[31:25]	RESERVED	Reserved. Read as "0"	RO	0	0
[24]	WATCHDOG	WatchDog Timer Reset Enable 0: WatchDog Timer reset triggers a PCI interrupt to be sent to the PCI host. 1: WatchDog Timer reset triggers a Soft reset which is equivalent to asserting IXP_RESET_0[16] (RST_ALL).	RW	1	0
[23]	Reserved	Reserved.	RW	1	1
[22]	EXT_RESET_EN	External Reset Enable (IXP2800 Only) If RESET_OUT_STRAP is "1" and this bit is: 0: The nRESET_OUT pin behaves the same as the internal reset caused by a hardware reset (reset is deasserted when the PLL lock is obtained). 1: Software controls the nRESET_OUT pin by writing to IXP_RESET_0[15] (EX_RST) Else if RESET_OUT_STRAP is "0", Software always controls the nRESET_OUT pin by writing to IXP_RESET_0[15] (EX_RST)	RW	0	0
[21]	INIT_COMP	Indicates that initialization is complete. 0— The IXP processor returns a retry response as the target of PCI configuration cycles, and will not assert PCI_DEV_SEL# to the PCI I/O or memory commands. 1—The IXP processor returns a normal response to PCI configuration cycles.	RW	0	0
[20]	CMD_ARB	Intel test bit. Users should not set it. For IXP2400, IXP2400 automatically sets this bit as part of system reset, and automatically clears it after 256 cycles. Command bus arbiter reset	RW	0	1



Bits	Field	Description	RW	IXP 28x0 Reset	IXP 2400 Reset
[19]	SBUS_ARB	Intel test bit. Users should not set it. For IXP2400, IXP2400 automatically sets this bit as part of system reset, and automatically clears it after 256 cycles. S Push/Pull Bus arbiter reset.	RW	0	1
[18]	DBUS_ARB	Intel test bit. Users should not set it. For IXP2400, IXP2400 automatically sets this bit as part of system reset, and automatically clears it after 256 cycles. D Push/Pull bus arbiter reset.	RW	0	1
[17]	SHaC	Intel test bit. Users should not set it. For IXP2400, IXP2400 automatically sets this bit as part of system reset, and automatically clears it after 256 cycles. Scratch Controller, Hash Unit and CSR unit reset.	RW	0	1
[16]	RST_ALL	Resets the chip in a manner equivalent to a Hard Reset. For IXP2400, IXP2400 automatically sets this bit as part of system reset, and automatically clears it after 256 cycles.	RW	0	1
[15]	EXT_RST	External Reset: If set, the external reset pin RESET_OUT# is asserted. Refer to IXP_RESET_0[22] (EXT_RESET_EN).	RW	0	1
[14]	DRAM_3	Not used: Reserved for future use.	RO	0	0
[13]	DRAM_2	Dram 2 Controller: Reserved on IXP2400.	RW	1	0
[12]	DRAM_1	Dram 1 Controller: Reserved on IXP2400.	RW	1	0
[11]	DRAM_0	Dram 0 Controller. For IXP2400, IXP2400 automatically clears this bit after 256 cycles as part of system reset.	RW	1	0
[10:8]	RESERVED	Reserved for MSF future use.	RO	0	0
[7]	MSF	Media and Switch Fabric Controller:	RW	1	1
[6]	SRAM_3	SRAM 3 Controller: IXP2800 only. Reserved on IXP2400.	RW	1	0
[5]	SRAM_2	SRAM 2 Controller: IXP2800 only. Reserved on IXP2400.	RW	1	0
[4]	SRAM_1	SRAM 1 Controller For IXP2400, IXP2400 automatically clears this bit after 256 cycles as part of system reset.	RW	1	0
[3]	SRAM_0	SRAM 0 Controller For IXP2400, IXP2400 automatically clears this bit after 256 cycles as part of system reset.	RW	1	0



Bits	Field	Description	RW	IXP 28x0 Reset	IXP 2400 Reset
[2]	PCI_RST	<p>If the CFG_RST_DIR pin is asserted high, PCI_RST# is an output and if this bit is:</p> <p>0 - IXP2800 / IXP2400 does not assert PCI_RST# pin.</p> <p>1 - IXP2800 / IXP2400 asserts PCI_RST# pin.</p> <p>The Intel XScale® core must clear this bit after reset to release the PCI bus from reset.</p> <p>If the CFG_RST_DIR pin is asserted low, PCI_RST# is an input. This bit will be asserted when PCI_RST# is asserted and cleared otherwise</p>	RW	1	1
[1]	PCI	PCI Unit: Should only be set as part of system reset	RW	1	1
[0]	XSCALE	<p>Intel XScale® core Reset: If the CFG_PROM_BOOT pin is:</p> <p>1: This bit is automatically cleared by hardware after the reset so that the Intel XScale® core can boot</p> <p>0: This bit remains high after reset and must be cleared by the PCI host driver after boot code has been loaded into DRAM. When it is deasserted, the Intel XScale® core will fetch the boot code from DRAM.</p>	RW	Refer to the description.	

5.6.4.6 CLOCK_CONTROL

Clock Control selects the clock ratio for the SRAM and DRAM controllers. This register must be programmed before accesses to the SRAM/DRAM are done. For the IXP2800, a value of 0x0 stops the clock and 0x1 and 0x2 are illegal.

For IXP2400 only, the clock control settings are documented in the description column of the following table.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVE D				APB_CLK_ RATIO				MSF_CLK_ _RATIO				DRAM_ CLK_ RATIO				SRAM_ CH3_CLK_ RATIO				SRAM_ CH2_CLK_ RATIO				SRAM_ CH1_CLK_ RATIO				SRAM_ CH0_CLK_ RATIO				

Bits	Field	Description	RW	IXP 2800 Reset	IXP 2400 Reset
[31:28]	RESERVED	Reserved	RO	0x0	0x0
[27:24]	APB_CLK_RATIO	IXP2800: Clock ratio for the Intel XScale® core peripheral devices. The frequency is 1/4n the ME frequency where n is the value in this field. 0x3 - 1/12 ME Frequency 0x4 - 1/16 ME Frequency ...0xf - 1/60 ME Frequency a value of 0 will disable the clock, which is not recommended due to it shutting off the SHaC unit. IXP2400: Reserved	RW	0xF	0x0
[23:20]	MSF_CLK_RATIO	IXP2800: Clock ratio for media and switch fabric interface if RCLK input clock supplied by the PHY interface is not used. MSF freq = (ME freq) / value in this field IXP2400: Reserved	RW	0xF	0x0
[19:16]	DRAM_CLK_RATIO	Clock ratio for all the DRAM Channels. IXP2800: DRAM internal clock frequency = (ME freq) / value in this field. The reference output clock (RefClk) is this frequency/2 and the CTM clock is 4x this frequency. IXP2400: DRAM freq = ME freq / value in this field. Valid values are 0x4 and 0x6.	RW	0xF	0x6
[15:12]	SRAM_CH3_CLK_RATIO	IXP2800: Clock ratio for SRAM Channel 3. An integer that used to derive the SRAM bus frequency. SRAM freq = (ME freq) / value in this field IXP2400: Reserved	RW	0xF	0x0

Bits	Field	Description	RW	IXP 2800 Reset	IXP 2400 Reset
[11:8]	SRAM CHANNEL 2 CLOCK RATIO	IXP2800: Clock ratio for SRAM Channel 2. An integer that used to derive the SRAM bus frequency. SRAM freq = (ME freq) / value in this field. IXP2400: Reserved	RW	0xF	0x0
[7:4]	SRAM CHANNEL 1 CLOCK RATIO	Clock ratio for SRAM Channel 1. An integer that used to derive the SRAM bus frequency. XP2800: SRAM freq = (ME freq) / value in this field. IXP2400: SRAM freq = ME freq / value in this field. Valid values are 0x3, 0x4, and 0x6.	RW	0xF	0x6
[3:0]	SRAM CHANNEL 0 CLOCK RATIO	Clock ratio for SRAM Channel 0. An integer that used to derive the SRAM bus frequency. IXP2800: SRAM freq = (ME freq) / value in this field. IXP2400: SRAM freq = ME freq / value in this field. Valid values are 0x3, 0x4, and 0x6.	RW	0xF	0x6

When the Clock Control register is programmed, typically through the integrated Intel XScale® processor or PCI interface, the IXP2400 network processor needs to resynchronize with the new clock ratio settings. User should allow adequate time for the synchronization to take effect. Specifically, software should allow 500 Intel XScale® core clock cycles or 100 PCI cycles to ensure that the synchronization is complete.

Here is a sample Intel XScale® code sequence for programming Clock_Control:

```
//Drain instruction
mcr    p15, 0, r0, c7, c10, 4
//Align the following code sequence to 32-byte cache
//line boundary.
.align 5
//Program Clock_Control register.
//Register r10 holds the address of Clock_Control.
//Register r2 holds the value to be programmed into Clock_Control.
str    r2, [r10]
//Perform a wait loop of about 500 or 0x200 cycles.
mov    r3, #0x200
1:
subs   r3, r3, #1
bne    1b
```

5.6.4.7 STRAP_OPTIONS

This register provides software visibility to board strapping options (i.e. pins that are tied high or low on the PC board to configure start-up options).

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED															RESET_OUT_STRAP	CFG_PLL_MULT					CFG_PCI_SWIN		CFG_PCI_DWIN	CFG_PCI_ARB	CFG_PROM_BOOT	CFG_RST_DIR				

Bits	Field	Description	RW	Reset
[31:15]	RESERVED	Reserved	RO	0
[14]	RESET_OUT_STRAP	IXP2800: The SP_AD[7] pins selects when the external reset pin (NRESET_OUT) is deasserted and the current strapping settings are provided in this field. 1: the reset deasserts after PLL Lock based on the default value from IXP_RESET_0[22] (EXT_RESET_EN which defaults to 0) 0: the reset deasserts under Software control by writing to IXP_RESET_0[15] (EX_RST). Refer to Section 5.6.4.4 IXP2400: Reserved	RO	dep
[13:8]	CFG_PLL_MULT	IXP2800: The SP_AD[5:0] pins selects IXP PLL Multiplier and the current strapping settings are provided in this field. Valid values are even values between (and including) 0x10 (16) to 0x30 (48). IXP2400: Reserved	RO	dep
[7:6]	CFG_PCI_SWIN	The GPIO[6:5] pins selects PCI SRAM BAR Window Size and the current strapping settings are provided in this field. 11—256 MByte 10—128 MByte 01—64 MByte 00— 32MByte	RO	dep
[5:4]	CFG_PCI_DWIN	The GPIO[4:3] pins selects PCI DRAM BAR Window Size and the current strapping settings are provided in this field. 11—1024 MByte 10—512 MByte 01—256 MByte 00—128 MByte	RO	dep
[3]	CFG_PCI_ARB	The GPIO[2] pin selects PCI Arbiter configuration and the current strapping settings are provided in this field. 0 — Uses an external arbiter 1 — Uses the internal arbiter	RO	dep

5.6.5 Timer (CAP CSR)

Table 5-29 shows the offset addresses of the Timer CSRs. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can be accessed by the Intel XScale® core, PCI and the MEs.

Note: In the IXP2800 Rev A, only timer register #1 and #4 are available. Timer registers #2 and #3 are reserved.

Table 5-29. Timer Register Map

Name	Abbreviation	Address	Description	Section
TIMER CONTROL registers	T1_CTL T2_CTL T3_CTL T4_CTL	0x00 0x04 0x08 0x0C	This is used to determine the timer functions, mode, activation	Section 5.6.5.1
TIMER COUNTER LOADING registers	T1_CLD T2_CLD T3_CLD T4_CLD	0x10 0x14 0x18 0x1C	These registers store the initial values for the timer counters. Writing to a register causes the timer to reload with its initial value.	Section 5.6.5.2
TIMER COUNTER STATUS register	T1_CSR T2_CSR T3_CSR T4_CSR	0x20 0x24 0x28 0x2C	This is to store the current counter values.	Section 5.6.5.3
TIMER COUNTER CLEAR registers	T1_CLR T2_CLR T3_CLR T4_CLR	0x30, 0x34, 0x38, 0x3C	Any write to these registers clear the associated timer interrupts.	Section 5.6.5.4
TIMER WATCHDOG ENABLE register	TWDE	0x40	This is to enable the timer 4 to be a watchdog timer.	Section 5.6.5.4

5.6.5.1 T#_CTL (# = 1,2,3,4)

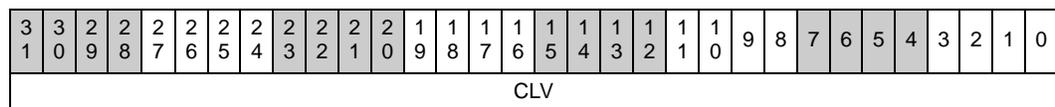
Timer Control Registers. This register controls the timer functions, mode, and activation.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																					ACT	RESERVED				PSS		RESERVED	

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved. Read returns 0; write has no effect	RO	0
[7]	ACT	Activate the Timer. When 1: Timer is activated. After the re-activation, the Timer Counter Status register value is resumed. When 0: Timer is deactivated. The counter is disabled but the value in the Timer Counter Status register can still be read.	RW	0x0
[6:4]	RESERVED	Reserved. Read returns 0; write has no effect	RO	0
[3:2]	PSS	Select the pre-scaler. The bit interpretation: 00: Use clock to trigger the counter; 01: Use clock/16 to trigger the counter; 10: Use clock/256 to trigger the counter; 11: Use GPIO pins to trigger the counter for external event. In this case, there are three limitations: <ul style="list-style-type: none"> The duration of the GPIO pulse has to be greater than 40ns The pulse delay has to be more than 40ns The maximum counter trigger frequency is 25MHz ideally. The GPIO Pins are assigned as follows: gpio[0] = timer1 gpio[1] = timer2 gpio[2] = timer3 gpio[3] = timer4.	RW	0x00
[1:0]	RESERVED	Reserved. Read returns 0; write has no effect	RO	0

5.6.5.2 T#_CLD, (# = 1,2,3,4)

Timer Counter Loader Register. These registers is used to hold the counter initial value. The value can be altered by the programmer. Writing to these registers causes the reload of the counters. By default, they contain the value of 0xFFFF_FFFF.



Bits	Field	Description	RW	Reset
[31:0]	CLV	Store the counter initial value.	RW	0xFFFFFFFF

5.6.5.3 T#_CSR, (# = 1,2,3,4)

Timer Counter Status Register. These registers store the current counter value. During reset, the T#_CSR registers get loaded with the initial value of the T#_CLD registers.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0							
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	CSV															

Bits	Field	Description	RW	Reset
[31:0]	CSV	Store the current counter value	RO	0xFFFFFFFF

5.6.5.4 T#_CLR(# = 1,2,3,4)

Timer (Interrupt) Clear Register. This is a set of four registers, one for each timer. Each of them has only one bit to be activated and used to clear corresponding interrupt signal, which are being asserted. It is a write-only register.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0								
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	RESERVED																ICL

Bits	Field	Description	RW	Reset
[31:1]	RESERVED	Reserved. Read returns undefined value in this case; write has no effect	RO	undef
[0]	ICL	When 0: No effect; When 1: Clear the interrupt being asserted	WO	0

5.6.5.5 TWDE

Timer Watchdog Enable Register. This register contains 1 bit, corresponding to timer 4. If it is set to one, the watchdog reset is enable; otherwise, it stays dormant. By default, it is set to 0. Once this bit is set it can only be cleared by a system reset.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0								
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	RESERVED																WDE

Bits	Field	Description	RW	Reset
[31:1]	RESERVED	Reserved. Read returns 0; write has no effect	RO	0
[0]	WDE	Watchdog Enable 0: Watchdog is disabled 1: Watchdog is enabled. Writing a one enables the watchdog function.	RW	0

5.6.6 GPIO (CAP CSR)

Table 5-30 shows the offset addresses of the GPIO CSRs. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can be accessed by the Intel XScale® core, PCI and the MEs.

The value of each GPIO pin can be read through the GPIO pin-level register (GPIO_PLR). This register can be read at any time, regardless of drive direction, and can be used to confirm the state of the pin. In addition, each GPIO pin can be programmed to detect a rising and/or falling edge through the GPIO rising-edge detect enable register (GPIO_REDR) and GPIO falling-edge detect enable register (GPIO_FEDR). The state of the edge detect can be read through the GPIO edge detect status register (GPIO_EDSR). These edge detects can be programmed to generate an interrupt. Also, each GPIO pin can be programmed to generate an interrupt on the state of the pin by enabling the level sensitive high and low detect register (GPIO_LSHR and GPIO_LSLR).

The outputs of the GPIO can be used to advance the timer counts in the Timer registers, when this feature is enabled. All of them are synchronized with the internal clock and converted to a pulse before feeding into the timer counters internally. GPIO[0] connects to Timer 0, GPIO[1] to Timer 1, GPIO[2] to Timer 2 and GPIO[3] to Timer 3.

Table 5-30. GPIO Register Map

Abbreviation	Address	Name	Description	Section
GPIO_PLR	0x00	GPIO Pin level register	This is used to determine the current value of a particular pin	Section 5.6.6.1
GPIO_PDPR	0x04	GPIO Pin direction programmable register	This is to program a pin as an input or a output	Section 5.6.6.2
GPIO_PDSR	0x08	GPIO Pin direction set register	This is to set a pin as an output	Section 5.6.6.3
GPIO_PDCR	0x0C	GPIO Pin direction clear register	This is to reset a pin as an input	Section 5.6.6.4
GPIO_POPR	0x10	GPIO Output data programmable register	This is to program the output data register	Section 5.6.6.5
GPIO_POSR	0x14	GPIO Output data set register	This is to set an output data register	Section 5.6.6.6
GPIO_POCR	0x18	GPIO Output data clear register	This is to clear an output data register	
GPIO_REDR	0x1C	GPIO Rising edge detect enable register	This is to enable detects on rising edge	Section 5.6.6.7
GPIO_FEDR	0x20	GPIO Falling edge detect enable register	This is to enable detects on falling edge	

Table 5-30. GPIO Register Map (Continued)

Abbreviation	Address	Name	Description	Section
GPIO_EDSR	0x24	GPIO Edge detect status register	This is the logging of detected transitions	Section 5.6.6.8
GPIO_LSHR	0x28	GPIO level sensitive high enable register	This is to enable detect on level sensitive high inputs.	Section 5.6.6.9
GPIO_LSLR	0x2C	GPIO level sensitive low enable register	This is to enable detect on level sensitive low inputs.	Section 5.6.6.9
GPIO_LDSR	0x30	GPIO level detect status register	This is to log the logic level of inputs.	Section 5.6.6.10
GPIO_INER	0x34	GPIO Interrupt Enable register	This is to enable the interrupt generation.	Section 5.6.6.11
GPIO_INSR	0x38	GPIO Interrupt Set register	This is to set the interrupt enable register.	Section 5.6.6.12
GPIO_INCR	0x3C	GPIO Interrupt Reset register	This is to reset the interrupt enable register.	Section 5.6.6.13
GPIO_INST	0x40	GPIO Interrupt Status Register	This is to capture the interrupts occurred to the corresponding pin by the external devices.	Section 5.6.6.14

5.6.6.1 GPIO_PLR

GPIO Pin Level Register. This is a read only register that is used to determine the current value of a particular pin (regardless of input/output). The state of each of the GPIO pins is visible through the GPIO pin-level register (GPIO_PLR). Each bit corresponds to the bit number. These are read-only registers that are used to determine the current value of a particular pin (regardless of the programmed pin direction).

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0										
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	RESERVED										PL							

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	PL	GPIO Pin level; write has no effect. 0—Pin state is low 1—Pin state is high	RO	0

In the following, two sets of registers are described. They are functionally quite similar, with each set consisting three registers. The user can use these registers to control either the direction of the pins or the data propagating through the pins. The first set contains GPIO_PDPR, GPIO_PDSR, and GPIO_PDCR. They allow the user to either program the direction of the pins by programming GPIO_PDPR, or set and reset the direction of the pins by using GPIO_PDSR and GPIO_PDCR. The main purpose is to provide the user a simple, flexible and efficient way to control the direction of these pins. For example, if the user wants to change most of the pin direction, they can use the GPIO_PDPR to program the direction of those pins. If only one or two pins need to be changed, the

user may simply use the GPIO_PDSR and GPIO_PDCR to set to output pins and reset to input pins. Applying the same approach, the user can also eliminate the usage of read-modify-write to alter the direction of a few pins.

Likewise, the second set of the registers contains three registers. They employ the same approach as described above for the first set of the direction control registers. However, these registers are used to control the pin output logic levels. GPIO_POPR allows the user to program each pin logic level while the GPIO_POSR and GPIO_POCR can set and reset the pin logic level respectively.

5.6.6.2 GPIO_PDPR

GPIO Pin Direction Programmable Register. Pin direction is controlled by programming the GPIO pin direction register. The GPIO_PDR registers contain one direction control bit for each of the 8 pins. If a direction bit is programmed to a 1, the port is an output. If it is programmed to a zero, it is an input. At reset, all bits in this register are cleared, configuring all GPIO pins as inputs. Reserved bits, should be written to zeros and reads to the reserved bits should be ignored.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
RESERVED																						PDP									

Note: For glitch free operation on GPIO outputs -- one has to write the pin output values first and then enable the pin as an output. Otherwise unknown values would be driven out on the GPIO pin.

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	PDP	GPIO Pin direction 0—Pin configured as an input 1—Pin configured as an output	RW	0

5.6.6.3 GPIO_PDSR

GPIO Pin Direction Set Register. This register consists of 8 bits, each corresponding to each pin of the GPIO. This register allow the programmer to set pin direction to be an output port simply by writing a one to register bit corresponding to that pin. However, there is no effect for writing a zero to register bits. Meanwhile, the value of the GPIO_PDPR is going to be updated accordingly. Reserved bits, should be written to zeros and reads to the reserved bits should be ignored.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
RESERVED																						PDS									

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	PDS	GPIO pin output direction set 0—Pin direction unaffected 1—Set pin to an output pin	WO	0

5.6.6.4 GPIO_PDCR

GPIO Pin Direction Clear Register. This register also has 8 bits associated with 8 GPIO. It is used for resetting the pin direction to be an input if a one is written to the corresponding register bit. It has no effect if it is programmed to zero. Similarly, the GPIO_PDPR is updated accordingly. Reserved bits, should be written to zeros and reads to the reserved bits should be ignored.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																				PDC											

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	PDC	GPIO pin input direction set 0—Pin direction unaffected 1—Clear pin to an input pin	WO	0

5.6.6.5 GPIO_POPR

GPIO Pin Output Programmable Register. Each GPIO pin is associated with each register bit. This register allows user to configure the output level of the pin. Reserved bits, should be written to zeros and reads to the reserved bits should be ignored

For glitch free operation on the GPIO output pins, the output values should be set prior to changing the GPIO pin direction, otherwise unknown values (in the case of the test a very short pulsed glitch) would appear on the pins..

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																				POP												

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	POP	GPIO output pin set 0—if pin configured as an output, set pin level low 1—if pin configured as an output, set pin level high	RW	0

5.6.6.6 GPIO_POSR, GPIO_POCR

GPIO Pin Output Set Register and GPIO Pin Output Clear Register

When a GPIO is configured as an output, the user controls the state of the pin by writing to either the GPIO pin output set registers (GPIO_POSR) or the GPIO pin output clear registers (GPIO_POCR). An output pin is set by writing a one to its corresponding bit within the GPIO_POSR. To clear an output pin, a one is written to the corresponding bit within the GPIO_POCR. These are write-only registers. Reads return unpredictable values.

Writing a zero to any of the GPIO_POSR or GPIO_POOCR bits has no effect on the state of the pin. Writing a one to a GPIO_POSR or GPIO_POOCR bit corresponding to a pin that is configured as an input will have effect only after the pin is configured as output. Writing a one to GPIO_POSR or GPIO_POOCR when the pin is configured as an input will register the value in the register. Reserved bits, must be written with zeros and reads should be ignored.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						PS									

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	PS	GPIO output pin set 0—Pin level unaffected 1—if pin configured as an output, set pin level high	WO	0

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						PC									

Bits	Field	Description	RW	Reset
[31:8]	reserved		RO	0
[7:0]	PC	GPIO output pin clear 0—Pin level unaffected 1—if pin configured as an output, clear pin level low	WO	0

5.6.6.7 GPIO_REDR, GPIO_FEDR

GPIO Rising Edge Detect Register and GPIO Falling Edge Detect Register. Each GPIO can also be programmed to detect a rising-edge, falling-edge, or either transition on a pin. It is required however, that the pulse following the edge be ATLEAST 100ns wide to guarantee a detection. When an edge is detected that matches the type of edges programmed for the pin, a status bit is set. The interrupt controller can be programmed to signal an interrupt to the CPU when any one of these status bits is set.

The GPIO rising-edge and falling-edge detect enable registers (GPIO_REDR and GPIO_FEDR, respectively) are used to select the type of transition on a GPIO pin that causes a bit within the GPIO edge detect enable status register (GPIO_EDSR) to be set. For a given GPIO pin, its corresponding GPIO_REDR bit is set to cause a GPIO_EDSR status bit to be set when the pin transitions from logic level zero (0) to logic level one (1). Likewise, GPIO_FEDR is used to set the corresponding GPIO_EDSR status bit when a transition from logic level one (1) to logic level zero (0) occurs. When the corresponding bits are set in both registers, either a falling- or a rising-edge transition causes the corresponding GPIO_EDSR status bit to be set.



3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						PRE								

Bits	Field	Description	RW	Reset
[31:8]	reserved		RO	0
[7:0]	PRE	GPIO pin rising edge detect enable 0—Disable rising edge detect enable 1—set corresponding GPIO_EDSR status bit when a rising edge is detected on the GPIO Pin	RW	0

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						PFE								

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	PFE	GPIO pin falling edge detect enable 0—Disable falling edge detect enable 1—set corresponding GPIO_EDSR status bit when a falling edge is detected on the GPIO Pin	RW	0

5.6.6.8 GPIO_EDSR

GPIO Edge Detect Status Register. The GPIO edge detect status register contains a total of 8 status bits that correspond to the 8 GPIO pins. When an edge detect occurs on a pin that matches the type of edge programmed in the GPIO_REDR and/or GPIO_FEDR registers, the corresponding status bit is set in GPIO_EDSR. Once a GPIO_EDSR bit is set, the CPU must clear it. GPIO_EDSR status bits are cleared by writing a one to them. Writing a zero to a GPIO_EDSR status bit has no effect.

Each edge detect that sets the corresponding GPIO_EDSR status bit for GPIO pins 0–7 can trigger an interrupt request. “Interrupt Controller” should have a description of the programming of GPIO interrupts.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						PSR								

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	PSR	GPIO edge detect status 0—No edge detect has occurred as specified in GPIO_REDR and GPIO_FEDR 1—Edge detect has occurred as specified in GPIO_REDR and GPIO_FEDR	RW 1C	0

5.6.6.9 GPIO_LSHR, GPIO_LSLR

GPIO Level Sensitive High (enable) Register and GPIO Level Sensitive Low (enable) Register. Each GPIO can be programmed to detect the level sensitive state of a pin. When the pin level matches the level detect enable programmed on the pin, a status bit is set. The interrupt controller can be programmed to signal an interrupt to the CPU when any of these status bits are set.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						PLH								

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	PLH	GPIO Level Sensitive High detect enable 0—Disable level sensitive high detect enable 1—set status bit in interrupts status register when a high state is detected on the GPIO pin.	RW	0

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						PLL								

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	PLL	GPIO Level Sensitive Low detect enable 0—Disable level sensitive low detect enable 1—set status bit in interrupts status register when a low state is detected on the GPIO pin.	RW	0

5.6.6.10 GPIO_LDSR

GPIO Level Detect Status Register. This register contains a total of 8 status bits that correspond to the 8 GPIO pins. When a level detect occurs on a pin that matches the type of level programmed in the GPIO_LSLR and/or GPIO_LSHR registers, the corresponding status bit is set in GPIO_LDSR. Once a GPIO_LDSR bit is set, the CPU must clear it. GPIO_LDSR status bits are cleared by writing a one to them. Writing a zero to a GPIO_LDSR status bit has no effect.

Each edge detect that sets the corresponding GPIO_LDSR status bit for GPIO pins 0–7 can trigger an interrupt request. “Interrupt Controller” should have a description of the programming of GPIO interrupts.

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	PSR	GPIO edge detect status 0—No level detect has occurred as specified in GPIO_LSLR and GPIO_LSHR 1—Level Detect has occurred as specified in GPIO_LSLR and GPIO_LSHR	RW 1C	0

5.6.6.11 GPIO_INER

GPIO Interrupt Enable Register. Each GPIO pin is associated with each register bit. This register allows user to configure the interrupt generation if an event occurs, like the edge or level detections of the corresponding pin. When setting a one, the corresponding pin interrupt is enabled and vice versa. Reserved bits, should be written to zeros and reads to the reserved bits should be ignored.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED														IER																	

Bits	Field	Description	RW	Reset
[31:8]	reserved		RO	0
[7:0]	IER	GPIO output pin set 0—Disable the interrupt with the corresponding pin; 1—Enable the interrupt with the corresponding pin.	RW	0

5.6.6.12 GPIO_INSR

GPIO Interrupt Enable Set Register. Each GPIO pin is associated with each register bit. This register allows user to set the interrupt enable register. When setting a one, the corresponding pin interrupt is enabled; otherwise, it keeps the same configuration as before.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED														IELR																	

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	IS	GPIO output pin set 0—No effect; 1—Enable the interrupt enable register with the corresponding pin.	WO	0

5.6.6.13 GPIO_INCR

GPIO Interrupt Enable Clear Register. Each GPIO pin is associated with each register bit. This register allows user to set the interrupt enable register. When setting a one, the corresponding pin interrupt is disabled; otherwise, it keeps the same configuration as before.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						IC									

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	IC	Interrupt Enable Registers. GPIO output pin set 0—No effect; 1—Disable the interrupt enable register with the corresponding pin.	WO	0

5.6.6.14 GPIO_INST

GPIO Interrupt Status Register. This register contains a total of 8 status bits that correspond to the 8 GPIO pins. When an interrupt occurs on a pin, the corresponding status bit is set in GPIO_INST. Once a GPIO_INST bit is set, the CPU must clear it. GPIO_INST status bits are cleared by writing a one to them. Writing a zero to a GPIO_INST status bit has no effect.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						IST									

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	IST	Interrupt Status Register. GPIO edge detect status 0—No interrupt has occurred; 1—Interrupt has occurred.	RW 1C	0

5.6.7 UART (CAP CSR)

Table 5-31 shows the offset addresses of the UART CSRs. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can be accessed by the Intel XScale® core, PCI and the MEs.

The UART supports registers similar to the 16550 UART. There are 12 registers in the UART. These registers share eight address locations in the I/O address space. Only for the lowest byte has data for all UART registers. Note that the state of the Divisor Latch Bit (DLAB), which is the MOST significant bit of the Serial Line Control Register, affects the selection of certain of the UART registers. The DLAB bit must be set high by the system software to access the Baud Rate Generator Divisor Latches.

Table 5-31. UART Register Map

Abbreviation	Address [7:0]	Name	Description	Section
UART_RBR	0x00, DLAB=0	UART Receive Buffer Register	It is used to buffer the received data.	Section 5.6.7.1
UART_THR	0x00, DLAB=0	UART Transmit Holding Register	It is used to hold the transmitting data.	Section 5.6.7.2
UART_DLRL	0x00, DLAB=1	UART Divisor Latch register Low	It is associated with UART_DLHR and used to control the baud rate together.	Section 5.6.7.3
UART_DLRH	0x04, DLAB=1	UART Divisor Latch Register High	It is associated with UART_DLRL and used to control the baud rate together.	Section 5.6.7.3
UART_IER	0x04, DLAB=0	UART Interrupt Enable Register	It is the interrupt enable register for all interrupt control.	Section 5.6.7.4
UART_IIR	0x08	UART Interrupt Identification Register	This is a read only register and shares the same space as UART_FCR	Section 5.6.7.5
UART_FCR	0x08	UART Fifo control register	This is a write only register. It is used to control the FIFO.	Section 5.6.7.6
UART_LCR	0x0C	UART Line Control Register	This is used to control the transmission line data format.	Section 5.6.7.7
UART_LSR	0x14	UART Line Status Register	This stores the status of the previous transaction.	Section 5.6.7.8
UART_SPR	0x1C	UART scratch pad register	This allows the program to access for programming purpose.	Section 5.6.7.9

5.6.7.1 UART_RBR

UART Receive Buffer Register. This register holds the characters received by the UART's receive shift register. If fewer than 8 bits are received, the bits are right justified and leading bits are zeroed. Reading the register empties the register and reset the Data Ready (DR) bit in the Line status register to 0. In FIFO mode this register latches the data byte at the bottom of the FIFO.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						RBR									

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	RBR	Receive Buffer Register. Data byte received, least significant bit first.	RO	0

5.6.7.2 UART_THR

UART Transmit Holding Register. This register holds the next data byte to be transmitted, When the Transmit Shift Register becomes empty, the contents of the Transmit Holding Register are loaded into the shift register and the transmit data request (TDRQ) bit in the Line Status Register is set to 1. In FIFO mode, writing to THR puts data at the top of the FIFO. The data at the bottom of the FIFO is loaded to the shift register when it is empty.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						THR										

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	THR	Data byte transmitted, least significant bit first	W0	0

5.6.7.3 UART_DLRL, UART_DLRH

Divisor Latch Registers. Each UART contains a programmable Baud Rate Generator that is capable of taking the fixed input clock and dividing it by any divisor from 2 to $(2^{16} - 1)$. The output frequency of the Baud Rate Generator is 16 times the baud rate. Two 8-bit latches store the divisor in a 16-bit binary format. These Divisor Latches must be loaded during initialization to ensure proper operation of the Baud Rate Generator. If both Divisor Latches are loaded with 0, the 16X output clock is stopped.

The baud rate of the data shifted in/out of a UART is given by:

$$\text{Baud Rate} = \text{APB Clock} / 16 \times \text{divisor}$$

A Divisor value of 0 in the Divisor Latch Register is not allowed. The reset value of the divisor is 02.



UART_DLRL

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						LB								

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	LB	Low byte for generating baud rate	RW	0x2

UART_DLRH

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						HB								

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	HB	High byte for generating baud rate	RW	0

5.6.7.4 UART_IER

UART Interrupt Enable Register. This register enables the four types of UART interrupts. Each interrupt can individually activate the interrupt (INTR) output signal. It is possible to totally disable the interrupt system by resetting bits 0 through 4 of the Interrupt Enable Register (UART_IER). Similarly, setting bits of the UART_IER register to a logic 1, enables the selected interrupt(s). Disabling an interrupt prevents it from being indicated as active in the IIR and from activating the INTR output signal. All other system functions operate in their normal manner, including the setting of the Line Status Registers.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						UUE	NRZE	RTOIE	RESERVED	RLSE	TIE	RAVIE		

Bits	Field	Description	RW	Reset
[31:7]	RESERVED	Reserved	RO	0
[6]	UUE	UART Unit Enable: 0—the UART unit is disabled 1—the UART unit is enabled	RW	0
[5]	NRZE	NRZ coding Enable: 0: NRZ coding disabled 1: NRZ coding enabled	RW	0
[4]	RTOIE	Receiver Time out Interrupt Enable: 0—Receiver data time out interrupt disabled 1—Receiver data time out interrupt enabled (Applicable only in FIFO mode and when UART_FCR ITL is set to a trigger level other than 1 byte.)	RW	0
[3]	RESERVED	Reserved	RW	0
[2]	RLSE	Receiver Line Status Interrupt Enable: 0—Receiver Line Status interrupt disabled 1—Receiver Line Status interrupt enabled	RW	0
[1]	TIE	Transmit Data request Interrupt Enable: 0—Transmit FIFO Data Request interrupt disabled 1—Transmit FIFO Data Request interrupt enabled	RW	0
[0]	RAVIE	Receiver Data Available Interrupt Enable: 0—Receiver Data Available (Trigger level reached) interrupt disabled 1—Receiver Data Available (Trigger level reached) interrupt enabled	RW	0

5.6.7.5 UART_IIR

UART Interrupt Identification Register. In order to minimize software overhead during data character transfers, the UART prioritizes interrupts into four levels (listed in Table 5-32) and records these in the Interrupt Identification register. The Interrupt Identification register (UART_IIR) stores information indicating that a prioritized interrupt is pending and the source of that interrupt.

Table 5-32. Interrupt Conditions

Priority Level	Interrupt Origin
1 (highest)	Receiver Line Status: one or more error bits were set.
2	Received Data is available. In FIFO mode, trigger level was reached; in non-FIFO mode, RBR has data.
2	Receiver Time out occurred. It happens only in FIFO mode and when the trigger level is set to 8, 16, or 32 bytes in the ITL field of the UART_FCR register. Specifically, it happens when some data that is less than the configured trigger level has been received in the receive FIFO, but there is no further activity for a time period.
3	Transmitter requests data. In FIFO mode, the transmit FIFO is half or more than half empty; in non-FIFO mode, THR is read already.
4 (lowest)	Not used (Modem status in standard UART)

If the CPU access is occurring at the same time when the UART records new interrupts, the UART doesn't pass the newly captured interrupts to the CPU until the current access has completed. Therefore, the CPU can only obtain the information of newly captured interrupts in the next access cycle.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						FIFOES		RESERVED		TOD	IID		IP#

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:6]	FIFOES[1:0]	FIFO Mode Enable Status: 00 = Non-FIFO mode is selected 01 = Reserved 10 = Reserved 11 = FIFO mode is selected (TRFIFOE = 1)	RO	0
[5:4]	RESERVED	Reserved	RO	0
[3]	TOD (IID3)	Time Out Detected: 0—No time out interrupt is pending 1—Time out interrupt is pending. (Only in FIFO mode and when UART_FCR ITL is set to a trigger level other than 1 byte.)	RO	0
[2:1]	IID (IID1,IID2)	Interrupt Source Encoded: 00 = NOT USED (In standard UART used to be Modem Status (CTS, DSR, RI, DCD modem signals changed state) 01 = Transmit FIFO requests data 10 = Received Data Available 11 = Receive error (Overrun, parity, framing, break, FIFO error)	RO	0
[0]	IP# (IID0)	Interrupt Pending: 0—Interrupt is pending. (Active low) 1—No interrupt is pending	RO	1

Table 5-33 shows how the above bits are to be used in decoding

Table 5-33. Interrupt Identification Register Decode

Interrupt ID bits UART_IIR[3:0]				Interrupt SET/RESET Function			
3	2	1	0	Priority	Type	Source	RESET Control
0	0	0	1	-	None	No Interrupt is pending.	-
0	1	1	0	Highest	Receiver Line Status	Overrun Error, Parity Error, Framing Error, Break Interrupt.	Reading the Line Status Register.

Table 5-33. Interrupt Identification Register Decode (Continued)

Interrupt ID bits UART_IIR[3:0]				Interrupt SET/RESET Function			
3	2	1	0	Priority	Type	Source	RESET Control
0	1	0	0	Second Highest	Received Data Available.	Non-FIFO mode: Receive Buffer is full.	Non-FIFO mode: Reading the Receiver Buffer Register.
						FIFO mode: Trigger level was reached.	FIFO mode: Reading bytes until Receiver FIFO drops below trigger level or setting RESETRF bit in FCR register.
1	1	0	0	Second Highest	Character Time-out indication.	FIFO Mode only: At least 1 character is in receiver FIFO and there was no activity for a time period.	Reading the Receiver FIFO or setting RESETRF bit in FCR register.
0	0	1	0	Third Highest	Transmit FIFO Data Request	Non-FIFO mode: Transmit Holding Register Empty	Reading the IIR Register (if the source of the interrupt) or writing into the Transmit Holding Register.
						FIFO mode: Transmit FIFO has half or less than half data.	Reading the IIR Register (if the source of the interrupt) or writing to the Transmitter FIFO.
0	0	0	0	Fourth Highest	NOT USED	NOT USED	NOT USED

5.6.7.6 UART_FCR

UART FIFO Control Register. UART_FCR is a write only register that is located at the same address as the UART_IIR (UART_IIR is a read only register). UART_FCR enables/disables the transmitter/receiver FIFOs, clears the transmitter/receiver FIFOs, and sets the receiver FIFO trigger level.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
RESERVED																						ITL		RESERVED		RESETRF	RESETRF	TRIFIFOE			

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:6]	ITL	<p>Interrupt Trigger Level: When the number of bytes in the receiver FIFO equals the interrupt trigger level programmed into this field and the Received Data Available Interrupt is enabled (via UART_IER), an interrupt is generated and appropriate bits are set in the UART_IIR.</p> <p>00 = 1 byte in FIFO causes interrupt 01 = 8 bytes in FIFO causes interrupt 10 = 16 bytes in FIFO causes interrupt 11 = 32 bytes in FIFO causes interrupt</p>	WO	0
[5:3]	RESERVED	Reserved	RO	0
[2]	RESETTF	<p>Reset transmitter FIFO: When RESETTF is set to 1, the transmitter FIFO counter logic is set to 0, effectively clearing all the bytes in the FIFO. The TDRQ bit in UART_LSR are set and IIR shows a transmitter requests data interrupt if the TIE bit in the IER register is set. The transmitter shift register is not cleared; it completes the current transmission. After the FIFO is cleared, RESETTF is automatically reset to 0.</p> <p>0—Writing 0 has no effect 1—The transmitter FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0</p>	WO	0
[1]	RESETRF	<p>Reset Receiver FIFO: When RESETRF is set to 1, the receiver FIFO counter is reset to 0, effectively clearing all the bytes in the FIFO. The DR bit in LSR is reset to 0. All the error bits in the FIFO and the FIFOE bit in LSR are cleared. Any error bits, OE, PE, FE or BI, that had been set in LSR are still set. The receiver shift register is not cleared. If IIR had been set to Received Data Available, it is cleared. After the FIFO is cleared, RESETRF is automatically reset to 0.</p> <p>0—Writing 0 has no effect 1—The receiver FIFO is cleared (FIFO counter set to 0). After clearing, bit is automatically reset to 0</p>	WO	0
[0]	TRFIFOE	<p>Transmit and Receive FIFO Enable: TRFIFOE enables/disables the transmitter and receiver FIFOs. When TRFIFOE = 1, both FIFOs are enabled (FIFO Mode). When TRFIFOE = 0, the FIFOs are both disabled (non-FIFO Mode). Writing a 0 to this bit clears all bytes in both FIFOs. When changing from FIFO mode to non-FIFO mode and vice versa, data is automatically cleared from the FIFOs. This bit must be 1 when other bits in this register are written or the other bits are not programmed.</p> <p>0—FIFOs are disabled 1—FIFOs are enabled</p>	WO	0

5.6.7.7 UART_LCR

UART Line Control Register. In the Line Control Register, the system programmer specifies the format of the asynchronous data communications exchange. The serial data format consists of a start bit (logic 0), five to eight data bits, an optional parity bit, and one or two stop bits (logic 1). The UART_LCR has bits for accessing the Divisor Latch and causing a break condition. The programmer can also read the contents of the Line Control Register. The read capability simplifies system programming and eliminates the need for separate storage in system memory.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	DLAB	SB	STKYP	EPS	PEN	STB	WLS
RESERVED																														

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7]	DLAB	Divisor register access bit: This bit is the Divisor Latch Access Bit. It must be set high (logic 1) to access the Divisor Latches of the Baud Rate Generator during a READ or WRITE operation. It must be set low (logic 0) to access the Receiver Buffer, the Transmit Holding Register, or the Interrupt Enable Register. 0—access Transmit Holding register (THR), Receive Buffer Register (RBR) and Interrupt Enable Register. 1—access Divisor Latch Registers (DLL and DLM)	RW	0
[6]	SB	Set break: This bit is the set break control bit. It causes a break condition to be transmitted to the receiving UART. When SB is set to a logic 1, the serial output (TXD) is forced to the spacing (logic 0) state and remains there until SB is set to a logic 0. This bit acts only on the TXD pin and has no effect on the transmitter logic. This feature enables the processor to alert a terminal in a computer communications system. If the following sequence is executed, no erroneous characters will be transmitted because of the break: Load 00H in the Transmit Holding register in response to a TDRQ interrupt After TDRQ goes high (indicating that 00H is being shifted out), set the break bit before the parity or stop bits reach the TXD pin Wait for the transmitter to be idle (TEMT = 1) and clear the break bit when normal transmission has to be restored During the break, the transmitter can be used as a character timer to accurately establish the break duration. In FIFO mode, wait for the transmitter to be idle (TEMP=1) to set and clear the break bit. 0—no effect on TXD output 1—forces TXD output to 0 (space)	RW	0

Bits	Field	Description	RW	Reset
[5]	STKYP	<p>Sticky Parity: This bit is the “sticky parity” bit, which can be used in multiprocessor communications. When PEN and STKYP are logic 1, the bit that is transmitted in the parity bit location (the bit just before the stop bit) is the complement of the EPS bit. If EPS is 0, then the bit at the parity bit location will be transmitted as a 1. In the receiver, if STKYP and PEN are 1, then the receiver compares the bit that is received in the parity bit location with the complement of the EPS bit. If the values being compared are not equal, the receiver sets the Parity Error bit in LSR and causes an error interrupt if line status interrupts were enabled. For example, if EPS is 0, the receiver expects the bit received at the parity bit location to be 1. If it is not, then the parity error bit is set. By forcing the bit value at the parity bit location, rather than calculating a parity value, a system with a master transmitter and multiple receivers can identify some transmitted characters as receiver addresses and the rest of the characters as data. If PEN = 0, STKYP is ignored.</p> <p>0—no effect on parity bit 1—Forces parity bit to be opposite of EPS bit value</p>	RW	0
[4]	EPS	<p>Even parity Select: This bit is the even parity select bit. When PEN is a logic 1 and EPS is a logic 0, an odd number of logic ones is transmitted or checked in the data word bits and the parity bit. When PEN is a logic 1 and EPS is a logic 1, an even number of logic ones is transmitted or checked in the data word bits and parity bit. If PEN = 0, EPS is ignored.</p> <p>0—sends or checks for odd parity 1—sends or checks for even parity</p>	RW	0
[3]	PEN	<p>Parity enable: This is the parity enable bit. When PEN is a logic 1, a parity bit is generated (transmit data) or checked (receive data) between the last data word bit and Stop bit of the serial data. (The parity bit is used to produce an even or odd number of ones when the data word bits and the parity bit are summed.)</p> <p>0—no parity function 1—allows parity generation and checking</p>	RW	0
[2]	STB	<p>Stop bits: This bit specifies the number of stop bits transmitted and received in each serial character. If STB is a logic 0, one stop bit is generated in the transmitted data. If STB is a logic 1 when a 5-bit word length is selected via bits 0 and 1, then 1 and one half stop bits are generated. If STB is a logic 1 when either a 6, 7, or 8-bit word is selected, then two stop bits are generated. The receiver checks the first stop bit only, regardless of the number of stop bits selected.</p> <p>0—1 stop bit 1—2 stop bits, except for 5-bit character then 1-1/2 bits</p>	RW	0
[1:0]	WLS	<p>Word Length select: The Word Length Select bits specify the number of data bits in each transmitted or received serial character.</p> <p>00 = 5-bit character (default) 01 = 6-bit character 10 = 7-bit character 11 = 8-bit character</p>	RW	0

5.6.7.8 UART_LSR

UART Line Status Register. This register provides status information to the processor concerning the data transfers. Bits 5 and 6 show information about the transmitter section. The rest of the bits contain information about the receiver.

In non-FIFO mode, three of the LSR register bits, parity error, framing error, and break interrupt, show the error status of the character that has just been received. In FIFO mode, these three bits of status are stored with each received character in the FIFO. LSR shows the status bits of the character at the top of the FIFO. When the character at the top of the FIFO has errors, the UART_LSR error bits are set and are not cleared until software reads UART_LSR, even if the character in the FIFO is read and a new character is now at the top of the FIFO.

Bits 1 through 4 are the error conditions that produce a receiver line status interrupt when any of the corresponding conditions are detected and the interrupt is enabled. These bits are not cleared by reading the erroneous byte from the FIFO or receive buffer. They are cleared only by reading UART_LSR. In FIFO mode, the line status interrupt occurs only when the erroneous byte is at the top of the FIFO. If the erroneous byte being received is not at the top of the FIFO, an interrupt is generated only after the previous bytes are read and the erroneous byte is moved to the top of the FIFO.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						FIFOE	TEMT	TDRQ	BI	FE	PE	OE	DR	

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7]	FIFOE	FIFO Error Status: In non-FIFO mode, this bit is 0. In FIFO Mode, FIFOE is set to 1 when there is at least one parity error, framing error, or break indication for any of the characters in the FIFO. Note that a processor read to the Line Status register does not reset this bit. FIFOE is reset when all erring bytes have been read from the Receive Buffer register. 0—No FIFO or no errors in receiver FIFO 1—At least one character in receiver FIFO has errors	RO	0
[6]	TEMT	Transmitter Empty: TEMT is set to a logic 1 when the Transmit Holding register and the Transmitter Shift register are both empty. It is reset to a logic 0 when either the Transmit Holding register or the transmitter shift register contains a data character. In FIFO mode, TEMT is set to 1 when the transmitter FIFO and the Transmit Shift register are both empty. 0—There is data in the Transmit Shift register, the holding register, or the FIFO 1—All the data in the transmitter has been shifted out	RO	1

Bits	Field	Description	RW	Reset
[5]	TDRQ	<p>Transmit Data Request: TDRQ indicates that the UART is ready to accept a new character for transmission. In addition, this bit causes the UART to issue an interrupt to the processor when the transmit data request interrupt enable is set high. The TDRQ bit is set to a logic 1 when a character is transferred from the Transmit Holding register into the Transmit Shift register. The bit is reset to logic 0 concurrently with the loading of the Transmit Holding register by the processor. In FIFO mode, TDRQ is set to 1 when half of the characters in the FIFO have been loaded into the Shift register or the RESETTF bit in FCR has been set to 1. It is cleared when the FIFO has more than half data. If more than 64 characters are loaded into the FIFO, the excess characters are lost.</p> <p>0—There is data in Holding register or FIFO waiting to be shifted out 1—Transmit FIFO has half or less than half data</p>	RO	1
[4]	BI	<p>Break Interrupt: BI is set to a logic 1 when the received data input is held in the spacing (logic 0) state for longer than a full word transmission time (that is, the total time of Start bit + data bits + parity bit + stop bits). The Break indicator is reset when the processor reads the Line Status Register. In FIFO mode, only one character (equal to 00H), is loaded into the FIFO regardless of the length of the break condition. BI shows the break condition for the character at the top of the FIFO, not the most recently received character.</p> <p>0—No break signal has been received 1—Break signal occurred</p>	RO	0
[3]	FE	<p>Framing Error: FE indicates that the received character did not have a valid stop bit. FE is set to a logic 1 when the bit following the last data bit or parity bit is detected as a logic 0 bit (spacing level). If the Line Control register had been set for two stop bit mode, the receiver does not check for a valid second stop bit. The FE indicator is reset when the processor reads the Line Status Register. The UART will resynchronize after a framing error. To do this it assumes that the framing error was due to the next start bit, so it samples this "start" bit twice and then takes in the "data". In FIFO mode, FE shows a framing error for the character at the top of the FIFO, not for the most recently received character.</p> <p>0—No Framing error 1—Invalid stop bit has been detected</p>	RO	0
[2]	PE	<p>Parity Error: PE indicates that the received data character does not have the correct even or odd parity, as selected by the even parity select bit. The PE is set to a logic 1 upon detection of a parity error and is reset to a logic 0 when the processor reads the Line Status register. In FIFO mode, PE shows a parity error for the character at the top of the FIFO, not the most recently received character.</p> <p>0—No Parity error 1—Parity error has occurred</p>	RO	0



Bits	Field	Description	RW	Reset
[1]	OE	<p>Overrun Error: In non-FIFO mode, OE indicates that data in the receiver buffer register was not read by the processor before the next character was transferred into the receiver buffer register, thereby destroying the previous character. In FIFO mode, OE indicates that all 64 bytes of the FIFO are full and the most recently received byte has been discarded. The OE indicator is set to a logic 1 upon detection of an overrun condition and reset when the processor reads the Line Status register.</p> <p>0—No data has been lost 1—Received data has been lost</p>	RO	0
[0]	DR	<p>Data Ready: Bit 0 is set to a logic 1 when a complete incoming character has been received and transferred into the receiver buffer register or the FIFO. In non-FIFO mode, DR is reset to 0 when the receive buffer is read. In FIFO mode, DR is reset to a logic 0 if the FIFO is empty (last character has been read from RBR) or the RESETRF bit is set in FCR.</p> <p>0—No data has been received 1—Data is available in RBR or the FIFO</p>	RO	0

5.6.7.9 UART_SPR

UART Scratchpad Register. This 8-bit Read/Write Register does not control the UART in anyway. It is intended as a scratchpad register to be used by the programmer to hold data temporarily.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																				SCRATCH										

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	SCRATCH	No effect on UART functionality	RW	0

5.6.8 PMU (Performance Monitor UNit) (CAP CSR)

Table 5-34 shows the offset addresses of the PMU CSRs. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can be accessed by the Intel XScale® core, PCI and the MEs.

Table 5-34. PMU Register Summary

CSR name	Offset	Description	Section
PMUCONTCFG	0x0F00	PMU Control Bus Configuration Register	Section 5.6.8.1
PMUSTAT	0x0E00	PMU Counter Interrupt status Register	Section 5.6.8.2
PMUMASK	0x0D00	PMU Counter Interrupt Mask Register	Section 5.6.8.3
PMUINTEN	0x0C00	PMU Counter Interrupt enable Register	Section 5.6.8.4
CHAPCMD0	0x0000	CHAP Counter 0 Command Register	Section 5.6.8.5
CHAPCMD1	0x0010	CHAP Counter 1 Command Register	
CHAPCMD2	0x0020	CHAP Counter 2 Command Register	
CHAPCMD3	0x0030	CHAP Counter 3 Command Register	
CHAPCMD4	0x0040	CHAP Counter 4 Command Register	
CHAPCMD5	0x0050	CHAP Counter 5 Command Register	
CHAPEVN0	0x0004	CHAP Counter 0 Event Register	Section 5.6.8.6
CHAPEVN1	0x0014	CHAP Counter 1 Event Register	
CHAPEVN2	0x0024	CHAP Counter 2 Event Register	
CHAPEVN3	0x0034	CHAP Counter 3 Event Register	
CHAPEVN4	0x0044	CHAP Counter 4 Event Register	
CHAPEVN5	0x0054	CHAP Counter 5 Event Register	
CHAPSTAT0	0x0008	CHAP Counter 0 status Register	Section 5.6.8.7
CHAPSTAT1	0x0018	CHAP Counter 1 status Register	
CHAPSTAT2	0x0028	CHAP Counter 2 status Register	
CHAPSTAT3	0x0038	CHAP Counter 3 status Register	
CHAPSTAT4	0x0048	CHAP Counter 4 status Register	
CHAPSTAT5	0x0058	CHAP Counter 5 status Register	

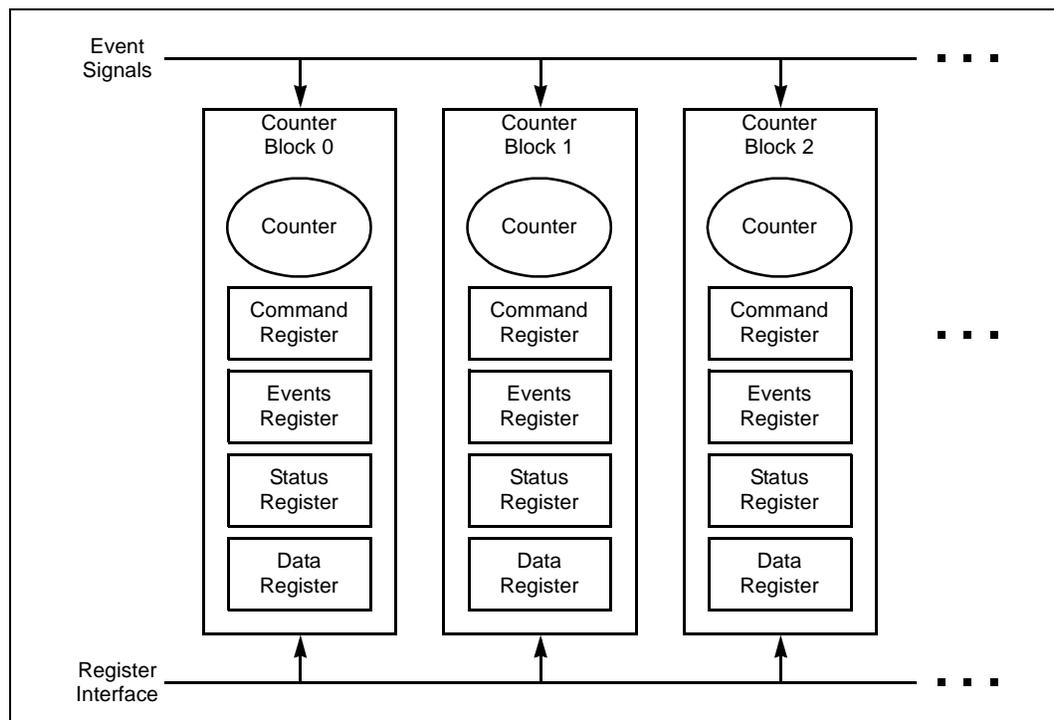
Table 5-34. PMU Register Summary (Continued)

CSR name	Offset	Description	Section
CHAPDATA0	0x000C	CHAP Counter 0 Data Register	Section 5.6.8.8
CHAPDATA1	0x001C	CHAP Counter 1 Data Register	
CHAPDATA2	0x002C	CHAP Counter 2 Data Register	
CHAPDATA3	0x003C	CHAP Counter 3 Data Register	
CHAPDATA4	0x004C	CHAP Counter 4 Data Register	
CHAPDATA5	0x005C	CHAP Counter 5 Data Register	

Because the CHAP unit resides on the APB bus, the offset associated with each of these registers is relative to the Memory Base Address that configuration software will set in the PMUADR register.

Each counter has one command, one event, one status, and one data register associated with it. Each counter is “packaged” with these four registers in a “counter block”. Each implementation selects the number of counters it will implement, and therefore how many counter blocks (or slices) it will have. These registers are numbered 0 through N - 1 where N represents the number of counters - 1. See [Figure 5-1](#).

Figure 5-1. Conceptual Diagram of Counter Array



5.6.8.1 PMUCONTCFG—PMU Control Bus Configuration Register

PMU Control Block consists of logic and state machine to generate the protocol on the Control bus and to generate mux selects for the PMU mux selects. Collectively this block programs all the design units and the PMU mux control block to route the right event to the counter chosen by software. Software programs the PMU Control Config register with appropriate event code and the PMU Control Bus State Machine generates Control Bus cycles to program the Design Units. Upon completion of the Control Bus cycle PMU control bus state machine generates a configuration done signal which can be monitored by the software to program the next set of counters.

PMU Control Block has two registers which can be written and read from APB bus. They are PMU Control Configuration registers and PMU Control Status Registers. PMU Control Configuration register PMUCONTCFG is programmed by the software to generate Control Bus cycles. A write to this register triggers the PMU Control Bus State Machine. Based on the command in the PMUCONTCFG PMU Control Bus State Machine generates the appropriate command on the Control Bus.

Software has to indicate to which counter the event from the design block is being routed. It also needs to indicate in the PMUCONTCFG register whether event being routed is to increment the counter, decrement the counter or to trigger the counter. Software also needs to indicate the mux in the design unit which is being used to route the event to the PMU. The 12 bit Event Selection Code indicates the event and the design unit where from the event is routed. PMUCONTCFG[6:0] indicates the event being routed while PMUCONTCFG[12:7] indicates the target design block id. PMU state machine supports 4 commands for routing and programming the performance monitoring muxes.

Note that IXP2400 and IXP2800 network processors only support the AUTOCONFIG command. Usage of the other commands by software results in undefined behavior.

Following are the actions by the PMU Control State Machine based on the command initiated by the software

Table 5-35. PMU Control Bus data Map

Command	Description	PMU Control Bus Configuration Register [31:0]	Target_Control_Bus [10:0]
Idle	Idle	Command [17:16] =00	[10:9] <= [17:16]
Reset	1. PMU Event bus bit Selection (total6) 2.Target ID Selection	Target Command [17:16] =01 PMU Event bus [15:13] Target ID [12:7]	[10:9] <= [17:16] [8:6] <= [15:13] [5:0] <= [12:7]
INIT	1. PMU Event bus bit Selection (total6) 2.Target ID Selection	Target Command [17:16] =01 PMU Event bus [15:13] Target ID [12:7]	[10:9] <= [17:16] [8:6] <= [15:13] [5:0] <= [12:7]
CONFIG	Event Selection (total 128 events)	Target Command [17:16] =11 Target Events [6:0]	[10:9] <= [17:16] [6:0] <= [6:0]

RESET:

PMU State Machine asserts Target_Control_Bus_active and puts RESET command on Target_Control_Bus[10:9] and puts target ID (PMUCONTCFG[12:7] of Target Design ID) on Target_Control_Bus[5:0] and inserts PMU Event bus on Target_Control_Bus[8:6]. The state machine in the design unit decodes the block number and if it is for that design uses the PMU Event bus and resets the mux selects for that mux.

INIT:

PMU State Machine asserts Target_Control_Bus_active and puts INIT command on Target_Control_Bus[10:9] and puts block ID (PMUCONTCFG[11:7] of Event Selection Code) on Target_Control_Bus[4:0] and inserts PMU Event bus on Target_Control_Bus[7:5]. The state machine in the design unit decodes the block number and if it is for that design uses the mux number and waits for config command for that mux. Software could choose to either use RESET command if it intends to clear the previous INIT command or could generate CONFIG cycle if it intends to configure the mux in the design unit.

PMU State Machine also generates selects to program the PMU mux control logic to route the event signals from the appropriate PMU Event bus (i.e of and 1 of the 6 design muxes) and selects the appropriate design block (one of the several design blocks) and it does this selection to the counter number defined in the PMUCONTCFG register and to one of the three event mux controls in each counter namely, increment, decrement or command trigger PMU Event bus. There are total of 18 mux controls in the PMU control block with three mux controls for each counter to select the increment, decrement and trigger events for that counter. There are 6 events generated from each design unit and there are over 10 design units generating those events. To make the right selection the PMU control state machine during the INIT cycle asserts one of the 6 Cntr_inc_sel and 6 Cntr_dec_sel and 6 Cntr_trig_sel signals. PMU control State Machine also puts the 3 bit mux number and 5 bit design block number in the PMUCONTCFG register on Cntr_mux_no and Cntr_design_blk signals that are inputs to PMU mux control block.

CONFIG:

This command should follow a INIT cycle. PMU State Machine asserts Target_Control_Bus_active and puts CONFIG command on Target_Control_Bus[10:9] and puts event number (PMUCONTCFG[6:0] of Design Event) on Target_Control_Bus[6:0]. The state machine in the design unit uses the event number and programs the mux that has been identified during the INIT cycle.

AUTOCONFIG:

This commands performs RESET followed by a INIT command followed by a CONFIG command. Upon completion of the control bus cycles it generates a Config done command which is registered in the PMUCONTSTS register.



3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED													AUTOCFG	RESERVED	SELECT_EVENT_BUS	SELECT_DESIGN_BLOCK	SELECT_DESIGN_EVENT														

Bits	Field	Description	RW	Reset
[31:19]	RESERVED	Reserved	RO	0
[18]	AUTOCFG	AUTOCFG command (IXP2400 and IXP2800 only support this command.) It will generate atomic operation. It will start from RESET, INIT, followed by CONFIG cycles to the design blocks. Upon completion it sets PMUCONTST[31]	RW	0
[17:16]	RESERVED	Reserved	RW	0
[15:13]	SELECT_EVENT_BUS	Select one out of 6 Event Bus which will route from design block to the counter.	RW	0
[12:7]	SELECT_DESIGN_BLOCK	Select one out of 35 design blocks.	RW	0
[6:0]	SELECT_DESIGN_EVENT	Select one out of 128 design events.	RW	0

5.6.8.2 PMUSTAT—PMU Counter Interrupt Status Registers

To clear any of the PMUSTAT interrupts and status bits, software should write a 0 to the respective status bits of the PMUSTAT register.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
TCD	RESERVED													CHAP_CNTR_5_STATUS	RESERVED	CHAP_CNTR_4_STATUS	RESERVED	CHAP_CNTR_3_STATUS	RESERVED	CHAP_CNTR_2_STATUS	RESERVED	CHAP_CNTR_1_STATUS	RESERVED	CHAP_CNTR_0_STATUS							

Bits	Field	Description	RW	Reset
[31]	TCD	PMU Target Design Configuration command was executed. When set, indicates Target Design Configuration command was executed.	RW	0
[30:23]	RESERVED		RW	0
[22:20]	CHAP_CNTR_5_STATUS	CHAP Counter 5 Status. This counter can generate 3 possible status conditions, which are reported through these bits in the status register. bit[20] When set, indicates Over Flow on Counter 5. bit[21] When set, indicates Command got Triggered on Counter 5. bit[22] When set, indicates Threshold Compare was observed on Counter 5.	RW	0
[19]	RESERVED		RW	0
[18:16]	CHAP_CNTR_4_STATUS	CHAP Counter 4 Status. This counter can generate 3 possible status conditions, which are reported through these bits in the status register. bit[16] When set, indicates Over Flow on Counter 4. bit[17] When set, indicates Command got Triggered on Counter 4. bit[18] When set, indicates Threshold Compare was observed on Counter 4.	RW	0
[15]	RESERVED		RW	0
[14:12]	CHAP_CNTR_3_STATUS	CHAP Counter 3 Status. This counter can generate 3 possible status conditions, which are reported through these bits in the status register. bit[12] When set, indicates Over Flow on Counter 3. bit[13] When set, indicates Command got Triggered on Counter 3. bit[14] When set, indicates Threshold Compare was observed on Counter 3.	RW	0
[11]	RESERVED		RW	0
[10:8]	CHAP_CNTR_2_STATUS	CHAP Counter 2 Status. This counter can generate 3 possible status conditions, which are reported through these bits in the status register. bit[8] When set, indicates Over Flow on Counter 2. bit[9] When set, indicates Command got Triggered on Counter 2. bit[10] When set, indicates Threshold Compare was observed on Counter 2.	RW	0
[7]	RESERVED		RW	



Bits	Field	Description	RW	Reset
[6:4]	CHAP_CNTR_1_STATUS	<p>CHAP Counter 1 Status. This counter can generate 3 possible status conditions, which are reported through these bits in the status register.</p> <p>bit[4] When set, indicates Over Flow on Counter 1.</p> <p>bit[5] When set, indicates Command got Triggered on Counter 1.</p> <p>bit[6] When set, indicates Threshold Compare was observed on Counter 1.</p>	RW	0
[3]	RESERVED		RW	0
[2:0]	CHAP_CNTR_0_STATUS	<p>CHAP Counter 0 Status. This counter can generate 3 possible status conditions, which are reported through these bits in the status register.</p> <p>bit[0] When set, indicates Over Flow on Counter 0.</p> <p>bit[1] When set, indicates Command got Triggered on Counter 0.</p> <p>bit[2] When set, indicates Threshold Compare was observed on Counter 0.</p>	RW	0

5.6.8.3 PMUMASK—PMU Counters Interrupt Mask Registers

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0				
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				
TCM		RESERVED										CHAP_CNTR_5_MASK		RESERVED		CHAP_CNTR_4_MASK		RESERVED		CHAP_CNTR_3_MASK		RESERVED		CHAP_CNTR_2_MASK		RESERVED		CHAP_CNTR_1_MASK		RESERVED		CHAP_CNTR_0_MASK	

Bits	Field	Description	RW	Reset
[31]	TCM	Target Configuration Mask. PMUMASK [31] masks PMUSTAT [31]. If PMUMASK [31] is set to 1, then PMUSTAT [31] does not register the status of the target design configuration command was executed or not.	RW	0
[30:23]	RESERVED	Reserved	RW	0
[22:20]	CHAP_CNTR_5_MASK	<p>CHAP Counter 5 Mask.</p> <p>Each of the CHAP counters can generate three possible status conditions, which can be masked by writing a 1 to the appropriate bit in the Mask register.</p> <p>PMUMASK [20] masks PMUSTAT [20]. If PMUMASK [20] is set to 1, then PMUSTAT [20] does not register the status of status condition0 in CHAP counter 5.</p> <p>PMUMASK [21] masks PMUSTAT [21]. If PMUMASK [21] is set to 1, then PMUSTAT [21] does not register the status of status condition1 in CHAP counter 5.</p> <p>PMUMASK [22] masks PMUSTAT [22]. If PMUMASK [22] is set to 1, then PMUSTAT [22] does not register the status of status condition2 in CHAP counter 5.</p>	RW	0
[19]	RESERVED	Reserved	RW	0
[18:16]	CHAP_CNTR_4_MASK	<p>CHAP Counter 4 Mask.</p> <p>Each of the CHAP counters can generate three possible status conditions, which can be masked by writing a 1 to the appropriate bit in the Mask register.</p> <p>PMUMASK [16] masks PMUSTAT [16]. If PMUMASK [16] is set to 1, then PMUSTAT [16] does not register the status of status condition0 in CHAP counter 4.</p> <p>PMUMASK [17] masks PMUSTAT [17]. If PMUMASK [17] is set to 1, then PMUSTAT [17] does not register the status of status condition1 in CHAP counter 4.</p> <p>PMUMASK [18] masks PMUSTAT [18]. If PMUMASK [18] is set to 1, then PMUSTAT [18] does not register the status of status condition2 in CHAP counter 4.</p>	RW	0
[15]	RESERVED	Reserved	RW	0

Bits	Field	Description	RW	Reset
[14:12]	CHAP_CNTRR_3_MASK	<p>CHAP Counter 3 Mask.</p> <p>Each of the CHAP counters can generate three possible status conditions, which can be masked by writing a 1 to the appropriate bit in the Mask register.</p> <p>PMUMASK [12] masks PMUSTAT [12]. If PMUMASK [12] is set to 1, then PMUSTAT [12] does not register the status of status condition0 in CHAP counter 3.</p> <p>PMUMASK [13] masks PMUSTAT [13]. If PMUMASK [13] is set to 1, then PMUSTAT [13] does not register the status of status condition1 in CHAP counter 3.</p> <p>PMUMASK [12] masks PMUSTAT [12]. If PMUMASK [12] is set to 1, then PMUSTAT [12] does not register the status of status condition2 in CHAP counter 3.</p>	RW	0
[11]	RESERVED	Reserved	RW	0
[10:8]	CHAP_CNTR_2_MASK	<p>CHAP Counter 2 Mask.</p> <p>Each of the CHAP counters can generate three possible status conditions, which can be masked by writing a 1 to the appropriate bit in the Mask register.</p> <p>PMUMASK [8] masks PMUSTAT [8]. If PMUMASK [8] is set to 1, then PMUSTAT [8] does not register the status of status condition0 in CHAP counter 2.</p> <p>PMUMASK [9] masks PMUSTAT [9]. If PMUMASK [9] is set to 1, then PMUSTAT [9] does not register the status of status condition1 in CHAP counter 2.</p> <p>PMUMASK [10] masks PMUSTAT [10]. If PMUMASK [10] is set to 1, then PMUSTAT [10] does not register the status of status condition2 in CHAP counter 2.</p>	RW	0
[7]	RESERVED	Reserved	RW	0

Bits	Field	Description	RW	Reset
[6:4]	CHAP_CNTR_1_MASK	<p>CHAP Counter 1 Mask.</p> <p>Each of the CHAP counters can generate three possible status conditions, which can be masked by writing a 1 to the appropriate bit in the Mask register.</p> <p>PMUMASK [4] masks PMUSTAT [4]. If PMUMASK [4] is set to 1, then PMUSTAT [4] does not register the status of status condition0 in CHAP counter 1.</p> <p>PMUMASK [5] masks PMUSTAT [5]. If PMUMASK [5] is set to 1, then PMUSTAT [5] does not register the status of status condition1 in CHAP counter 1.</p> <p>PMUMASK [6] masks PMUSTAT [6]. If PMUMASK [6] is set to 1, then PMUSTAT [6] does not register the status of status condition2 in CHAP counter 1.</p>	RW	0
[3]	RESERVED	Reserved	RW	0
[2:0]	CHAP_CNTR_0_MASK	<p>CHAP Counter 0 Mask.</p> <p>Each of the CHAP counters can generate three possible status conditions, which can be masked by writing a 1 to the appropriate bit in the Mask register.</p> <p>PMUMASK [0] masks PMUSTAT [0]. If PMUMASK [0] is set to 1, then PMUSTAT [0] does not register the status of status condition0 in CHAP counter 0.</p> <p>PMUMASK [1] masks PMUSTAT [1]. If PMUMASK [1] is set to 1, then PMUSTAT [1] does not register the status of status condition1 in CHAP counter 0.</p> <p>PMUMASK [2] masks PMUSTAT [2]. If PMUMASK [2] is set to 1, then PMUSTAT [2] does not register the status of status condition2 in CHAP counter 0.</p>	RW	0



5.6.8.4 PMUINTEN—PMU Interrupt Enable Register

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0					
TCE		RESERVED										CHAP_CNTR_5_INT_ENABLE		RESERVED		CHAP_CNTR_4_INT_ENABLE		RESERVED		CHAP_CNTR_3_INT_ENABLE		RESERVED		CHAP_CNTR_2_INT_ENABLE		RESERVED		CHAP_CNTR_1_INT_ENABLE		RESERVED		CHAP_CNTR_0_INT_ENABLE	

Bits	Field	Description	RW	Reset
[31]	TCE	<p>PMU Target design command Execute Interrupt Enable</p> <p>If PMUSTAT [31] is set to 1, then to generate an interrupt to XSCALE TCE (PMUINTEN [31]) needs to be programmed to 1.</p> <p>All interrupts signals from several different CHAP counters are ORed together to generate single interrupt to the PMU software interface.</p>	RW	0
[30:23]	RESERVED	Reserved	RW	0
[22:20]	CHAP_CNTR_5_INTERRUPT_ENABLE	<p>CHAP Counter 5 Interrupt Enable</p> <p>Each of the CHAP counters can generate 3 possible status conditions, which can be enabled to generate a interrupt by writing a 1 to the appropriate bit in the Interrupt enable register. This register supports 6 counters.</p> <p>If PMUSTAT [20] is set to 1, then to generate an interrupt to XSCALE PMUINTEN [20] needs to be programmed to 1.</p> <p>If PMUSTAT [21] is set to 1, then to generate an interrupt to XSCALE PMUINTEN [21] needs to be programmed to 1.</p> <p>If PMUSTAT [22] is set to 1, then to generate an interrupt to XSCALE PMUINTEN [22] needs to be programmed to 1.</p> <p>All interrupts signals from several different CHAP counters are ORed together to generate single interrupt to the PMU software interface.</p>	RW	0
[19]	RESERVED	Reserved	RW	0

Bits	Field	Description	RW	Reset
[18:16]	CHAP_CNTR_4_INTERRUPT_ENABLE	CHAP Counter 4 Interrupt Enable. This counter can generate 3 possible status conditions, which can be enabled to generate a interrupt by writing a 1 to the appropriate bit in the Interrupt enable register.	RW	0
[15]	RESERVED	Reserved	RW	0
[14:12]	CHAP_CNTR_3_INTERRUPT_ENABLE	CHAP Counter 3 Interrupt Enable. This counter can generate 3 possible status conditions, which can be enabled to generate a interrupt by writing a 1 to the appropriate bit in the Interrupt enable register.	RW	0
[11]	RESERVED	Reserved	RW	0
[10:8]	CHAP_CNTR_2_INTERRUPT_ENABLE	CHAP Counter 2 Interrupt Enable. This counter can generate 3 possible status conditions, which can be enabled to generate a interrupt by writing a 1 to the appropriate bit in the Interrupt enable register.	RW	0
[7]	RESERVED	Reserved	RW	0
[6:4]	CHAP_CNTR_1_INTERRUPT_ENABLE	CHAP Counter 1 Interrupt Enable. This counter can generate 3 possible status conditions, which can be enabled to generate a interrupt by writing a 1 to the appropriate bit in the Interrupt enable register.	RW	0
[3]	RESERVED	Reserved	RW	0
[2:0]	CHAP_CNTR_0_INTERRUPT_ENABLE	CHAP Counter 0 Interrupt Enable. This counter can generate 3 possible status conditions, which can be enabled to generate a interrupt by writing a 1 to the appropriate bit in the Interrupt enable register.	RW	0

5.6.8.5 CHAPCMDN—CHAP Command N Register (N = 0..5)

This 32-bit register allows control of the CHAP counter N. When this register is written, the previous register contents are overwritten. All 32 bits must be programmed each time the register is written. If the register contained a command that was still waiting to be triggered, it would be flushed without ever being executed. The currently executing command will continue to be executed only until the newly programmed command is triggered to execute.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0						
DBG	RESERVED					OUIE		CTIE		THIE		CONDITION CODE				SAC	OPCODE				RESERVED		CTEMS		CTDBS			RESERVED							ACT

Table 5-36. CHAP Command N Register Bit Definition (Sheet 1 of 4)

Bits	Field	Description	RW	Reset
[31]	DBG	<p>PMU DEBUG mode</p> <p>[31]=0 Normal PMU function.</p> <p>[31]=1 Each CHAP counter the event signals map to increment0, increment1, decrement0, decrement1 and trigger event will keep in CHAP status register[20:16] instead of go to CHAP counter.</p> <p>[20] keep increment0 event</p> <p>[19] keep increment1 event</p> <p>[18] keep decrement0 event</p> <p>[17] keep decrement1 event</p> <p>[16] keep trigger event</p>	RW	0
[30:27]	RESERVED	Reserved	RW	0
[26]	OUIE	<p>Overflow/Underflow Indicator Enable (OUIE)</p> <p>0 = No indication provided when a counter overflow or underflow occurs except for setting the Overflow/Underflow Indicator (OUI) status bit.</p> <p>1 = CHAP output signal will also be asserted to indicate that a counter overflow or underflow occurs. This is an output enable bit that is shared by all indicators. A global interrupt mask controls whether an interrupt is also generated. An interrupt service routine can check the status bits associated with enabled interrupts to determine how the interrupt was generated.</p>	RW	0
[25]	CTIE	<p>Command Trigger Indicator Enable (CTIE)</p> <p>0 = No indication provided when a command is triggered except for setting the Command Trigger Indicator (CTI) status bit.</p> <p>1 = CHAP output signal will also be asserted when a command is triggered. This is an output signal that is shared by all indicators. A global interrupt mask controls whether an interrupt is also generated. An interrupt service routine can check the status bits associated with enabled interrupts to determine how the interrupt was generated.</p>	RW	0

Table 5-36. CHAP Command N Register Bit Definition (Sheet 2 of 4)

Bits	Field	Description	RW	Reset
[24]	THIE	<p>Threshold Indicator Enable (THIE)</p> <p>0 = No indication provided when threshold condition is true except for setting the Threshold Indicator (TI) status bit</p> <p>1 = CHAP output signal will also be asserted when threshold condition is true. This is an output signal that is shared by all indicators. A global interrupt mask controls whether an interrupt is also generated. An interrupt service routine can check the status bits associated with enabled interrupts to determine how the interrupt was generated.</p>	RW	0
[23:21]	CC	<p>Condition Code (CC)</p> <p>This field contains the code that indicates what type of threshold compare is done between the counter and the data register. For all non-0 values of this field, the counter's data register will contain the threshold value. The outcome of this compare will generate a threshold event and potentially an interrupt if that capability is enabled.</p> <p>Bit 23 is for less than (<) Bit 22 is for equal (=) Bit 21 is for greater than (>)</p> <p>Select the proper bit mask for desired threshold condition: 000 = False (no threshold compare) 001 = Greater Than 010 = Equal 011 = Greater Than or Equal 100 = Less Than 101 = Not Equal 110 = Less Than or Equal 111 = True (always generate threshold event)</p>	RW	0
[20]	SAC	<p>Select ALL Counters (SAC)</p> <p>0 = The instruction is applied only to the counter associated with this command register.</p> <p>1 = The instruction is applied to ALL counters. This means that every command register is written to with the same value that was written to this particular command register. This is particularly handy for resetting all counters with a single command or starting or stopping all counters simultaneously.</p> <p>This bit is only valid in Command Register 0 and should be reserved in all non-0 command registers.</p> <p>"Globally" executed commands (by setting this bit in Command Register 0) always override "locally" executed commands.</p>	RW	0

Table 5-36. CHAP Command N Register Bit Definition (Sheet 3 of 4)

Bits	Field	Description	RW	Reset
[19:16]	OPCODE	<p>Opcode</p> <p>0000 = Stop. The corresponding counter does not count.</p> <p>0001 = Start. The corresponding counter begins counting. Each counter increments by one if the corresponding increment event occurs or decrements by one if the corresponding decrement event occurs. All duration type events toggle every CHAP unit clock tick that the event is true. The desired increment and decrement events must be selected before this command executes.</p> <p>0010 = Sample. The corresponding counter value is latched into the corresponding data register, which can then be read by reading the appropriate data register. The counter continues to count without being reset.</p> <p>0100 = Reset. The corresponding counter and register is reset to 0000 0000h. The 32 bit wide data registers allow 4 billion clock ticks or occurrences to be counted between sample commands. When the counter rolls over, the overflow status bit will be set in the corresponding status register.</p> <p>0101 = Restart. The corresponding counter resets, then starts counting again. This is essentially a Reset & Start command. This functionality will facilitate histogramming by allowing an event to trigger to clear the counter and resume counting with no further intervention.</p> <p>0110 = Sample & Restart. The Sample command happens and is followed immediately by the Restart command.</p> <p>1111 = Preload. The corresponding counter is set to the value that is located in the associated data register. This facilitates rollover and overflow validation. The counter remains in the same state when preloaded. If the counter was counting before the preload was executed it will continue to count after the preload. It is software's responsibility to ensure that the counter is in the desired state (example: execute stop command) prior to issuing a preload command.</p> <p>All others reserved.</p>	RW	0
[15]	RESERVED	Reserved	RW	0
[14:12]	CTEMS	<p>Command Trigger Event Mux Select (CTEMS)</p> <p>This field contains the Event Selection.</p> <p>Select one out of six events from each design group block to each associated counter to trigger</p>	RW	0
[11:8]	CTDBS	<p>Command Trigger Design Block Select (CTDBS)</p> <p>This field contains the Design group block trigger Selection.</p> <p>Select one out of 10 design group blocks to each associated counter command trigger.</p> <p>This field contains the PMU design group blocks that the unit will be required to detect before executing the opcode. The previously programmed opcode continues to execute until this command trigger is detected.</p>	RW	0
[7:1]	RESERVED	Reserved	RW	0

Table 5-36. CHAP Command N Register Bit Definition (Sheet 4 of 4)

Bits	Field	Description	RW	Reset
[0]	ACT	Always Command Trigger (ACT) 1= This causes the command to be triggered immediately upon being written to the command register. 0= This causes the command to be triggered by the command trigger [11:7] command register.	RW	0

5.6.8.6 CHAPEVN—CHAP Events N Register (N = 0..5)

This 32-bit register contains the events that control the increment and decrement of CHAP counter N. When this register is written, the previous register contents are overwritten (the event fields are unbuffered) and the associated counter will immediately be affected by the change. This register should only be programmed when the counter is idle and before the command register receives an opcode that is associated with the events in this register. If both an increment and a decrement event are detected on the same clock cycle, the counter value will not change.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	
DOCE		DEC1_MUX_SEL		RESERVED		DEC0_MUX_SEL		RESERVED		DDB		IOCE		INC1_MUX_SEL		RESERVED		INC0_MUX_SEL		RESERVED		IDB									

Table 5-37. CHAP Events N Register Bit Definition

Bits	Field Name	Description	RW	RESET
[31]	DOCE	Decrement Occurrence Count Enable (DOCE) 0 = Decrement Duration Count : the counter is decrement for each clock for which the decrement event signal is asserted logic high 1 = Decrement Occurrence Count : the counter is decrement each time a rising edge of the decrement event signal is detected.	RW	0
[30:28]	DEC1_MUX_SEL	Dec1 Mux Sel This field contains the Event Selection. Select one out of six events from each design group block to each associated counter Decrement 1 Select 0-5 are valid. Select 6-8 are disable the DEC1_MUX_SEL	RW	0
[27:26]	RESERVED	Reserved	RW	0
[25:23]	DEC0_MUX_SEL	Dec0 Mux Sel This field contains the Event Selection. Select one out of six events from each design group block to each associated counter Decrement 0 Select 0-5 are valid. Select 6-8 are disable the DEC0_MUX_SEL.	RW	0

Table 5-37. CHAP Events N Register Bit Definition (Continued)

Bits	Field Name	Description	RW	RESET
[22:20]	RESERVED	Reserved	RO	0
[19:16]	DDBS	Decrement Design Block Selection Select one out of 10 Design group blocks to counter Decrement 0 and Decrement 1 In case of event block are in 2x of PMU frequency domain then Counter may need two Decrement otherwise select either one of them.	RW	0
[15]	IOCE	Increment Occurrence Count Enable (IOCE) 0 = Increment Duration Count : the counter is increment for each clock for which the increment event signal is asserted logic high 1 = Increment Occurrence Count : the counter is decrement each time a rising edge of the increment event signal is detected.	RW	0
[14:12]	INC1_MUX_SEL	Inc1 Mux Sel This field contains the Event Selection. Select one out of six events from each design group block to each associated counter Increment 1 Select 0~5 are valid. Select 6~8 are disable the INC1_MUX_SEL.	RW	0
[11:10]	RESERVED	Reserved	RW	0
[9:7]	INC0_MUX_SEL	Inc0 Mux Sel This field contains the Event Selection. Select one out of six events from each design group block to each associated counter Increment 0 Select 0~5 are valid. Select 6~8 are disable the INC0_MUX_SEL.	RW	0
6:4	RESERVED	Reserved	RW	0
3:0	IDBS	Increment Design Block Selection Select one out of 10 Design group blocks to counter Decrement 0 and Decrement 1 In case of event block are in 2x of PMU frequency domain then Counter may need two Decrements otherwise select either one of them.	RW	0

Figure 5-2. Count Types Example

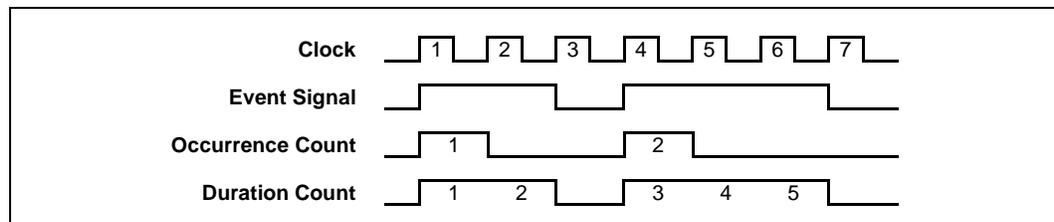


Table 5-38. CHAP Status N Register Bit Definition (Continued)

Bit No	Field Name	Description	RW	RESET
[25]	CTI	Command Trigger Indicator (CTI), WC 0 = NO commands have been triggered since the last time this bit was cleared. 1 = A command WAS triggered since the last time this bit was cleared. Software can use this bit to determine whether a command that was pending earlier has been triggered. Once a command has been triggered, another command can be triggered to execute.	RW1C	0
[24]	THI	Threshold Indicator (THI), WC 0 = No threshold event has been generated since the last time this bit was cleared. 1 = This counter generated a threshold event due to a true threshold condition compare since the last time this bit was cleared.	RW1C	0
[23:21]	RESERVED	Reserved	RO	0
[20:16]	DME	Debug Mode Event. If PMU Control bus configuration Register bit[31] set to 1 (debug mode), then CHAP counter all the events (increment0, increment1, decrement0, decrement1 and trigger event) will keep in these CHAP Status register[20:16], it will keep update every CPP clock. [20] keep increment0 event [19] keep increment1 event [18] keep decrement0 event [17] keep decrement1 event [16] keep trigger event	RO	0
[15:0]	RESERVED	Reserved	RO	0

5.6.8.8 CHAPDATAN—CHAP Data N Register (N = 0...5)

This 32-bit register allows for reading of the sampled value from CHAP event counter N. When write to this register it will keep the threshold value that will be compared to the value in the event counter when a threshold condition is in effect. Read and write will not go to same register. They share the same address, one for read only (sampled counter value) and the other one for the write only (threshold value).

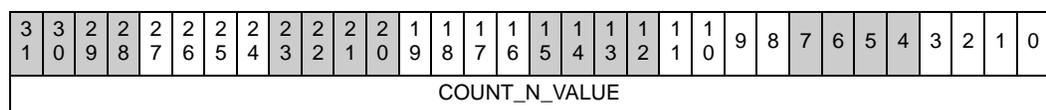


Table 5-39. CHAP Data N Register Bit Definition

Bits	Field Name	Description	RW	RESET
31:0	COUNTER_N_VALUE	Counter N Value: Contains either duration (number of clock ticks) or occurrences contained in CHAP event counter n at time of sampling. The register is programmed to contain the threshold value that will be compared to the value in the event counter when non-0 condition codes have been selected.	RW	0

5.6.9 SlowPort (CAP CSR)

Table 5-40 shows the offset addresses of the Slowport CSRs. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can be accessed the Intel XScale® core, PCI and the MEs.

Table 5-40. SlowPort Register Map

Abbreviation	Address	Name	Description	Section
SP_CCR	0x0000	Clock Configuration Register	This allows the user to configure the clock frequency at the SlowPort	Section 5.6.9.1
SP_WTC1 SP_WTC2	0x0004 0x0008	Write Timing Control Registers	These registers are used to control the timing of the waveforms for write access.	Section 5.6.9.2 Section 5.6.9.3
SP_RTC1 SP_RTC2	0x000C 0x0010	Read Timing Control Registers	These registers are used to control the timing of the waveforms for read access.	Section 5.6.9.4 Section 5.6.9.5
SP_FSR	0x0014	Fault Status Register	This register stores the previous transaction status.	Section 5.6.9.6
SP_PCR	0x0018	Protocol Control Register	This defines the protocol being used by the second SlowPort.	Section 5.6.9.7
SP_ADC	0x001C	Address Size Control Register	This defines the address and data widths.	Section 5.6.9.8
SP_FAC	0x0020	Flash Memory Address Size Register	This defines the address size of the flash memory used.	Section 5.6.9.9
SP_FRM	0x0024	Flash Memory Read Mode Register	This defines the data width read back from the flash memory.	Section 5.6.9.10
SP_FIN	0x0028	Framer Interrupt Enable Register	This enables the framer interrupt.	Section 5.6.9.11
SP_RXE	0x002C	Receive Enable Register	This enables configuration of input data sampling timing.	Section 5.6.9.13
SP_TXE	0x0030	Transmit Enable Register	This enables configuration of output data driving timing.	Section 5.6.9.12

5.6.9.1 SP_CCR

Clock Configuration Register. This register is used to allow the user to configure the output clock frequency driven onto SlowPort. It is going to be a 4-bit register so as to allow the user to specify the divisor from 1 up to 30. The default value is set to 0x0 with divisor of 1. All the reserved bits are read zero and have no effect for write access.

The IXP2xxx rev B enhances the flexibility of configuring the Slowport bus timing. As a result of this enhancement, the definition of the Divisor field of the SP_CCR register has changed. Both the original and the new definitions are captured in the following two tables. The original version applies to rev A of IXP2400 and IXP2800 network processors. The new version applies to rev B of IXP2400 and IXP2800 network processors.

In [Table 5-41](#), core circuit clock means the APB clock.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																								DIVISOR						

Bits	Field	Description	RW	Reset
[31:4]	RESERVED	Reserved. Read returns 0; write has no effect	RO	0
[3:0]	DIVISOR	Configure the clock divider. refer to Table 5-41 and Table 5-42	RW	0 (for rev A), 1 (for rev B)

Table 5-41. Corresponding Clock Division Values with Respect to the Register Values (for IXP2xxx rev A)

SP_CCR[3:0]	Divisor	Description
0000	0x01	Divide core circuit clock by 1
0001	0x02	Divide core circuit clock by 2
0010	0x04	Divide core circuit clock by 4
0011	0x06	Divide core circuit clock by 6
0100	0x08	Divide core circuit clock by 8
0101	0x0A	Divide core circuit clock by 10
0110	0x0C	Divide core circuit clock by 12
0111	0x0E	Divide core circuit clock by 14
1000	0x10	Divide core circuit clock by 16
1001	0x12	Divide core circuit clock by 18
1010	0x14	Divide core circuit clock by 20
1011	0x16	Divide core circuit clock by 22
1100	0x18	Divide core circuit clock by 24

Table 5-41. Corresponding Clock Division Values with Respect to the Register Values (for IXP2xxx rev A)

SP_CCR[3:0]	Divisor	Description
1101	0x1A	Divide core circuit clock by 26
1110	0x1C	Divide core circuit clock by 28
1111	0x1E	Divide core circuit clock by 30

Table 5-42. Corresponding Clock Division Values with Respect to the Register Values (for IXP2400 rev B)

SP_CCR[3:0]	Divisor	Description
0000	0x04	Divide internal bus clock by four. The period of the internal bus clock is two times the period of the micro-engine clock. For 400MHz parts, Divisor must be 0x06 or bigger.
0001	0x06	Divide internal bus clock by six
0010	0x08	Divide internal bus clock by eight
0011	0xA	Divide internal bus clock by ten
0100	0xC	Divide internal bus clock by twelve
0101	0xE	Divide internal bus clock by fourteen
0110	0x10	Divide internal bus clock by sixteen
0111	0x12	Divide internal bus clock by eighteen
1000	0x14	Divide internal bus clock by twenty
1001	0x16	Divide internal bus clock by twenty two
1010	0x18	Divide internal bus clock by twenty four
1011	0x1A	Divide internal bus clock by twenty six
1100	0x1C	Divide internal bus clock by twenty eight
1101	0x1E	Divide internal bus clock by thirty
1110	0x20	Divide internal bus clock by thirty two
1111	0x22	Divide internal bus clock by thirty four

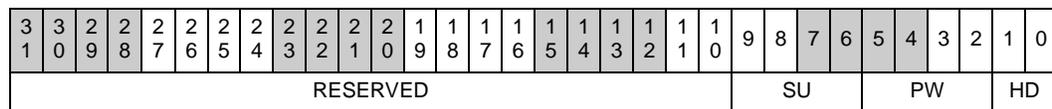
Table 5-43. Corresponding Clock Division Values with Respect to the Register Values (for

IXP2800 rev B)

SP_CCR[3:0]	Divisor	Description
0000	0x10	Divide internal bus clock by sixteen. The period of the internal bus clock is two times the period of the micro-engine clock.
0001	0x12	Divide internal bus clock by eighteen
0010	0x14	Divide internal bus clock by twenty
0011	0x16	Divide internal bus clock by twenty two
0100	0x18	Divide internal bus clock by twenty four
0101	0x1A	Divide internal bus clock by twenty six
0110	0x1C	Divide internal bus clock by twenty eight
0111	0x1E	Divide internal bus clock by thirty
1000	0x20	Divide internal bus clock by thirty two
1001	0x22	Divide internal bus clock by thirty four
1010	0x24	Divide internal bus clock by thirty six
1011	0x26	Divide internal bus clock by thirty eight
1100	0x28	Divide internal bus clock by forty
1101	0x2A	Divide internal bus clock by forty two
1110	0x2C	Divide internal bus clock by forty four
1111	0x2E	Divide internal bus clock by forty six

5.6.9.2 SP_WTC1

Write Timing Control Register for Device 1. There are two timing control registers, each corresponding to one device. They are used to control the access timing for setup time, pulse duration, and hold time. SP_WTC1 controls Flash PROM write parameters, while SP_WTC2 controls the second device write access on the Slow Port.



Bits	Field	Description	RW	Reset
[31:10]	Reserved	Read returns 0; write has no effect.	RO	0x0
[9:6]	SU	Delay from the address strobe to the data strobe; for mode 1 it represents the duration of the address strobe.	RW	0x1
[5:2]	PW	Pulse width for the data strobe, SP_WR_L; the value of this field should be greater than zero for fixed-timed device; otherwise, it will be treated as self-timing device with the SP_ACK_L activated. During the self-timing mode, the SP_ACK_L should respond after SU number of SP_CLK cycles.	RW	0xC

Bits	Field	Description	RW	Reset
[1:0]	HD	Hold time for the data from the data strobe to the end of cycle.	RW	0

5.6.9.3 SP_WTC2

Write Timing Control Register for Device 2. There are two timing control registers, each corresponding to one device. They are used to control the access timing for setup time, pulse duration, and hold time. SP_WTC1 controls Flash PROM write parameters, while SP_WTC2 controls the second device write access on the Slow Port.

This register has 10-bit width. The field definition is changed according to the mode set in the protocol control register (SP_PCR [Section 5.6.9.7](#)). In mode 0 (used for FlashROMs), the 4 bit SU field is allocated to the setup time control, indicating the number of clock cycles available from SP_CS_L assertion to the assertion of the SP_WR_L, depending on read or write access. The 4 bit PW field is allocated to the pulse width control. If this field is set to 0, the device will be treated as a self timing device; otherwise it is assumed to be a fixed timing and the number of clock cycles the pulse width spans is dictated by the value stored in this field. It indicates the number of clock cycles elapsed before the termination of the SP_WR_L signal. The 3 bit HD field is allocated to the hold delay timing control. It allows a certain number of clock cycles to pass before the termination of the transaction.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																				SU			PW			HD					

Bits	Field	Description	RW	Reset
[31:10]	Reserved	Read returns 0; write has no effect.	RO	0x0
[9:6]	SU	<p>Mode 0: Delay from the address strobe, SP_CS_L to the data strobe, SP_WR_L for write or SP_RD_L for read.</p> <p>Mode 1: Duration for SP_CS_L(CS); it represents the duration of whole transaction cycle;</p> <p>Mode 2: Number of SP_CLK cycle from the last data of data to the assertion of SP_CS_L and SP_WR_L/WRB for write;</p> <p>Mode 3: Delay from asserted SP_CS_L/CSB to asserted SP_WR_L/WRB for write;</p> <p>Mode 4: Delay from asserted SP_CS_L/CSB to asserted SP_RD_L/E; SP_WR_L/RWB is also asserted together with SP_CS_L;</p>	RW	0x1

Bits	Field	Description	RW	Reset
[5:2]	PW	<p>Mode 0: Pulse width for the data strobe, SP_WR_L; the value of this field should be greater than zero for fixed timed device; otherwise, it will be treated as self-timing device with the SP_ACK_L activated. During the self-timing mode, the SP_ACK_L should respond after SU number of SP_CLK cycles.</p> <p>Mode 1: Duration for SP_RD_L (R/W); the value of this field should be greater than zero but less than or equal to the value in the SU field described above for fixed-timed mode; if it is programmed to zero, it will be treated as self-timing mode. In the self-timing mode the ACK is expected after the SP_WR_L deasserted at least.</p> <p>Mode 2: Duration for SP_CS_L and SP_WR_L/WRB; this field cannot be set to zero; otherwise, it treats the transaction as self-timing transaction and the SP_ACK_L/INTB as an ACK signal. During the self-timing mode, the ACK is expected to appear after the number of SP_CLK cycle set in the SU field.</p> <p>Mode 3: Duration for SP_WR_L assertion for write; this field cannot be set to zero; otherwise, it treats the transaction as self-timing transaction and the SP_ACK_L/INTB as an ACK signal. During the self-timing mode, the ACK is expected to appear after the number of SP_CLK cycle set in the SU field.</p> <p>Mode 4: Duration for SP_RD_L/E assertion; this field cannot be set to zero; otherwise, it treats the transaction as self-timing transaction and the SP_ACK_L/INTB as an ACK signal. During the self-timing mode, the ACK is expected to appear after the number of SP_CLK cycle set in the SU field.</p>	RW	0x2
[1:0]	HD	<p>Mode 0: Hold time for the data from the data strobe to the end of cycle.</p> <p>Mode 1: Duration for SP_WR_L (ADS); the normal value of this field should be equal to 1;</p> <p>Mode 2: Number of SP_CLK cycle before the termination of the current transaction;</p> <p>Mode 3: Delay from deasserted SP_WR_L to deasserted SP_CS_L;</p> <p>Mode 4: Delay from deasserted SP_RD_L/E to deasserted SP_CS_L/CSB; SP_WR_L/RWB is also terminated together with SP_CS_L/E.</p>	RW	0

5.6.9.4 SP_RTC1

Slow Port Read Timing Control for Device 1. There are two timing control registers (SP_RTC1 and SP_RTC2), each corresponding to a slowport device. They are used to control the access timing for setup time, pulse duration, and hold time. SP_RTC1 controls the read parameters for the first device (typically Flash PROM), while SP_RTC2 controls the second device on the SlowPort.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																				SU			PW			HD					

Bits	Field	Description	RW	Reset
[31:10]	RESERVED	Reserved. Read returns 0; write has no effect.	RO	0x0
[9:6]	SU	SP_RTC1: Delay from the address strobe SP_CS_L assertion to the data strobe SP_RD_L assertion.	RW	0x1
[5:2]	PW	SP_RTC1: Pulse width for the data strobe, SP_RD_L assertion; the value of this field should be greater than zero for fixed-timed device; otherwise, it will be treated as self-timing device with the SP_ACK_L activated. The SP_ACK_L should appear after SU number of SP_CLK cycles during the self-timing mode.	RW	0xC
[1:0]	HD	SP_RTC1: Hold time for the data from the data strobe, SP_RD_L, deassertion to the end of cycle, SP_CS_L deassertion.	RW	0x0

5.6.9.5 SP_RTC2

Slow Port Read Timing Control for Device 2. There are two timing control registers (SP_RTC1 and SP_RTC2), each corresponding to a slowport device. They are used to control the access timing for setup time, pulse duration, and hold time. SP_RTC1 controls the read parameters for the first device (typically Flash PROM), while SP_RTC2 controls the second device on the SlowPort.

This register has 10-bit width. The field definition is changed according to the mode set in the protocol control register (SP_PCR [Section 5.6.9.7](#)). In mode 0 (used for FlashROMs), the 4 bit SU field is allocated to the delay control of the number of clock cycles available from SP_CS_L assertion to the assertion of the SP_RD_L. The 4 bits PW field is allocated to the pulse width control that indicates the number of clock cycles elapsed before the termination of the SP_RD_L signal. The 3 bit HD field is allocated to the delay number of clock cycles between the deassertion of SP_RD_L or SP_WR_L and the termination of the transaction. However, for mode 1 set in the protocol control register, the functionality of these fields are changed. SU, PW, and HD represent the number of clock cycles for the SP_CS (CS), SP_RD_L (RW), and SP_WR_L (ADS), respectively.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																				SU			PW			HD					

Bits	Field	Description	RW	Reset
[31:10]	RESERVED	Reserved. Read returns 0; write has no effect.	RO	0x0
[9:6]	SU	<p>Mode 0: Delay from the address strobe SP_CS_L assertion to the data strobe SP_RD_L assertion.</p> <p>Mode 1: For Fixed timing the setup value should be equal to or greater than the pulse width value. The setup value should not be set to zero (invalid entry)</p> <p>Mode 2: Number of SP_CLK cycle from the de-assertion of SP_ALE_L to the assertion of SP_CS_L and SP_RD_L/RDB;</p> <p>Mode 3: Delay from asserted SP_CS_L/CSB to asserted SP_RD_L/RDB;</p> <p>Mode 4: Delay from asserted SP_CS_L/CSB to asserted SP_RD_L/E;</p>	RW	0x1
[5:2]	PW	<p>Mode 0: Pulse width for the data strobe, SP_RD_L assertion; the value of this field should be greater than zero for fixed-timed device; otherwise, it will be treated as self timing device with the SP_ACK_L activated. The SP_ACK_L should appear after SU number of SP_CLK cycles during the self-timing mode.</p> <p>Mode 1: Duration for SP_RD_L (R/W); the value of this field should be greater than zero but less than or equal to the value in the SU field described above for fixed-timed mode; if it is programmed to zero, it will be treated as self-timing mode. During the self-timing mode, the ACK should appear after the SP_WR_L is deasserted.</p> <p>Mode 2: Duration for SP_CS_L and SP_RD_L/RDB for read; this field cannot be set to zero; otherwise, it treats the transaction as self-timing and the SP_ACK_L/INTB as an ACK signal. During self-timing mode, the SP_ACK_L is expected after the SU number of SP_CLK cycles.</p> <p>Mode 3: Duration for SP_RD_L; this field cannot be set to zero; otherwise, it treats the transaction as self-timing and the SP_ACK_L/INT as an ACK signal. During self-timing mode, the SP_ACK_L should appear after the SU number of SP_CLK cycles.</p> <p>Mode 4: Duration for SP_RD_L/E assertion; this field cannot be set to zero; otherwise, it treats the transaction as self-timing and the SP_ACK_L/INT as an ACK signal. During self-timing mode, the SP_ACK_L should appear after the SU number of SP_CLK cycles.</p>	RW	0x2

Bits	Field	Description	RW	Reset
[1:0]	HD	<p>Mode 0: Hold time for the data from the data strobe, SP_RD_L, deassertion to the end of cycle, SP_CS_L deassertion.</p> <p>Mode 1: Duration for SP_WR_L (ADS); the normal value of this field should be equal to 1 and should be at least one SP_CLK cycle less than SP_CS_L;</p> <p>Mode 2: Number of SP_CLK cycles before the termination of the current transaction; usually it is zero;</p> <p>Mode 3: Delay from de-asserted SP_RD_L to de-asserted SP_CS_L;</p> <p>Mode 4: Delay from de-asserted SP_RD_L/E to de-asserted SP_CS_L/CSB.</p>	RW	0x0

5.6.9.6 SP_FSR

Fault Status Register. The SP_FSR reports the time-out error if the transaction cycle is terminated before the SP_ACK_L returns back. The transaction is limited to 256 clock cycle of the SP_CLK(not the SHXP_APB_CLK). The SP_ACK_L must respond back within 256 clock cycle of the SP_CLK(not the SHXP_APB_CLK). This only happens in the self-timing device. Afterwards, an interrupt signal is issued back to the bus master. The interrupt is cleared by writing 0x1 to the FIN field.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	
RESERVED																											FIN	TOW2	TOR2	TOW1	TOR1

Bits	Field	Description	RW	Reset
[31:5]	RESERVED	Reserved. Read returns 0; write has on effect	RO	0
[4]	FIN	0: No interrupt; 1: Interrupt from the external framer device 2	RW1C	0
[3]	TOW2	0: No time-out occurs; 1: Write transaction time-out for device 2	RW1C	0
[2]	TOR2	0: No time-out occurs; 1: Read transaction time-out for device 2	RW1C	0
[1]	TOW1	0: No time-out occurs; 1: Write transaction time-out for device 1	RW1C	0
[0]	TOR1	0: No time-out occurs; 1: Read transaction time-out for device 1	RW1C	0



5.6.9.7 SP_PCR

Protocol Control Register. Flash PROM interface uses the generic bus protocol (mode 0) on SlowPort. For the second device, user can configure the SlowPort for different application. The SP_PCR is equipped for the user to configure the SlowPort interface with the non-standardized SONET/SDH microprocessor interfaces. The type of the microprocessor interface is presented in the microprocessor interface configuration (MIC) field as shown below.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																											MIC				

Bits	Field	Description	RW	Reset	
[31:3]	RESERVED	Reserved. Read returns 0; write has no effect	RO	0	
[2:0]	MIC	Configure μ P interface type.	RW	0	
		000			Mode 0: use generic bus protocol
		001			Mode 1: designated for the device similar to Lucent TDAT042G5 SONET/SDH μ P interface
		010			Mode 2: designated for the device similar to PMC-Sierra PM5351 S/UNI-TETRA μ P interface
		011			Mode 3: designated for the device similar to Intel and AMCC SONET/SDH Intel μ P interface
100	Mode 4: designated for the device similar to Intel and AMCC SONET/SDH Motorola μ P interface				

5.6.9.8 SP_ADC

Address Size/Data Width Control Register. The SP_ADC is used to configure the SlowPort interface with the non-standardized SONET/SDH microprocessor interface. The address size can be specified by AS field and the data width, by the DW field.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																								DW	RESERVED	AS					

Bits	Field	Description	RW	Reset
[31:6]	RESERVED	Reserved. Read returns 0; write has no effect	RO	0x0
[5:4]	DW	Configure data width. For mode 0, value 0 is treated as single byte transfer; otherwise, it will be treated as 4-byte transaction. 00 - Use 8-bit data width 01 - Use 16-bit data width 10 - Use 24-bit data width (this case may not exist) 11 - Use 32-bit data width For mode 0, a value of zero is treated as a single byte transfer; otherwise it is treated as a 4-byte transaction.	RW	0x0
[3:2]	RESERVED	Reserved. Read returns 0; write has no effect	RO	0x0
[1:0]	AS	Configure address width. 00 - Use 8-bit address space 01 - Use 16-bit address space 10 - Use 24-bit address space 11 - Use 32-bit address space; actually the maximum space is 25 bits which is only used for the mode 1,2,3,4 since mode 0 uses two control pins for the lower address bit[1:0].	RW	0x2

5.6.9.9 SP_FAC

Flash Memory Address Size Control Register. The SP_FAC is used to configure the address size of the flash memory. The address size can be specified by FAS field.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																												FAS		

Bits	Field	Description	RW	Reset
[31:2]	RESERVED	Reserved. Read returns 0; write has on effect	RO	0
[1:0]	FAS	Configure flash memory 00 - Use 10-bit address space 01 - Use 18-bit address space 10 - Use 25-bit address space 11 - Reserved	RW	0x2

5.6.9.10 SP_FRM

Flash Read Mode Register. The SP_FRM is used to configure the SlowPort flash memory read mode. There are two read mode, 8-bit read mode and 32-bit read mode. For the 8-bit read mode, one read cycle is involved. No packing process is needed. The data will be directly placed onto the lower order byte, [7:0] of the APB bus. For the 32-bit read mode, it needs 4 read cycles. All 4 bytes are packed into a 32-bit data and passed to the APB bus. By default, it is configured to 32-bit mode. The programmer should leave it to be 32-bit read mode most of the time unless they need to program the flash memory and access registers inside the flash memory.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																												FRM		

Bits	Field	Description	RW	Reset
[31:1]	RESERVED	Reserved. Read returns 0; write has on effect	RO	0
[0]	FRM	Configure flash memory read mode 0 - Use 32-bit data 1 -Use 8-bit data	RW	0

5.6.9.11 SP_FIN

Framer Interrupt Enable Register. The SP_FIN allows the user to enable or disable the interrupt signal from the framer. Since the interrupt signal shares the same pin of the sp_ack_1, the user should disable the interrupt if the sp_ack_1 is used as an ack signal.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																												FIN			

Bits	Field	Description	RW	Reset
[31:1]	RESERVED	Reserved. Read returns 0; write has on effect	RO	0
[0]	FIN	Framer interrupt enable. 0 - Disable 1 - Enable	RW	0x0

5.6.9.12 SP_TXE

Transmit Enable Register. The SP_TXE allows the users to configure the timing of data written out at the Slowport interface by the network processor. Specifically, relative to the active edge of the Slowport clock, data gets written out after TXE+1 number of internal bus clock cycles. The period of the internal bus clock is two times the period of the micro-engine clock. The maximum value for TXE depends on the setting of the SP_CCR register, which controls the divisor between the internal bus clock and the Slowport clock. The maximum value for TXE is the SP_CCR Divisor field value minus 2.

This register applies to IXP2400 rev B only.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																								TXE						

Bits	Field	Description	RW	Reset
[31:6]	RESERVED	Reserved. Read returns 0; write has on effect	RO	0
[5:0]	TXE	TXE + 1 reflects the delay between active edge of Slowport clock and driving out Slowport data in number of internal bus clocks. Legal values for IXP2400 rev B includes 0 through the SP_CCR Divisor value minus 2. For instance, if Divisor is 6, the Slowport clock is 6 times slower than the internal bus clock and the maximum value for TXE is 6 - 2, equal 4. Other values result in undefined behavior.	RW	0x1

5.6.9.13 SP_RXE

Receive Enable Register. The SP_RXE allows the users to configure the timing of data being sampled at the Slowport interface by the network processor. Specifically, data gets sampled RXE+1 number of internal bus clock cycles before the following active edge of the Slowport clock. The period of the internal bus clock is two times the period of the micro-engine clock. The maximum value for RXE depends on the setting of the SP_CCR register, which controls the divisor between the internal bus clock and the Slowport clock. The maximum value for RXE is the SP_CCR Divisor field value minus 2.

This register applies to IXP2400 rev B only.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																								RXN						

Bits	Field	Description	RW	Reset
[31:6]	RESERVED	Reserved. Read returns 0; write has on effect	RO	0
[5:0]	RXE	RXE + 1 reflects the number of internal bus clocks before the next active edge of Slowport clock that input data gets sampled at the Slowport. Legal values for IXP2400 rev B includes 0 through the SP_CCR Divisor value minus 2. For instance, if Divisor is 6, the Slowport clock is 6 times slower than the internal bus clock and the maximum value for RXE is 6 - 2, equal 4. Other values result in undefined behavior.	RW	0x1

5.7 Media and Switch Fabric Interface (MSF) - IXP2800

Table 5-44 shows the offset addresses of the IXP2800 MSF CSRs. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can be accessed by the Intel XScale® core, PCI and the MEs.

Note: For IXP2800 Rev A only -- Before using MSF, write a ‘1’ to bit 10, and then write a ‘0’ to bit 10 for the following register addresses -- 0x80f4, 0x80f8, 0x80fc, 0x8100, 0x8104. These registers are only used for manufacturing test and must be initialized prior to use. Since there are other test bits in these registers, in order to write bit 2, the programmer should read the value of the register, modify bit 2, then write back the modified value of the register.

Table 5-44. MSF Register Summary (Sheet 1 of 5)

Register	Offset (Hex)	Comment	Section
MSF_RX_CONTROL	0x0000		Section 5.7.1
MSF_TX_CONTROL	0x0004		Section 5.7.2
MSF_INTERRUPT_STATUS	0x0008		Section 5.7.3
MSF_INTERRUPT_ENABLE	0x000C		Section 5.7.4
CSIX_TYPE_MAP	0x0010		Section 5.7.5
FC_EGRESS_STATUS	0x0014		Section 5.7.6
FC_INGRESS_STATUS	0x0018		Section 5.7.7
Reserved	0x001C		
Reserved	0x0020		
HWM_CONTROL	0x0024		Section 5.7.17
FC_STATUS_OVERRIDE	0x0028		Section 5.7.8
MSF_CLOCK_CONTROL	0x002C		Section 5.7.9
RX_THREAD_FREELIST_0	0x0030		Section 5.7.18
RX_THREAD_FREELIST_1	0x0034		
RX_THREAD_FREELIST_2	0x0038		
Reserved	0x003C	RX_THREAD_FREELIST_3 in IXP2400	
RX_PORT_MAP	0x0040		Section 5.7.19
RBUF_ELEMENT_DONE	0x0044		Section 5.7.20
RX_CALENDAR_LENGTH	0x0048	RX_MPHY_POLL_LIMIT in the IXP2400	Section 5.7.21
FCE_FIFO_VALIDATE	0x004C		Section 5.7.22
RX_THREAD_FREELIST_TIMEOUT_0	0x0050		Section 5.7.24
RX_THREAD_FREELIST_TIMEOUT_1	0x0054		
RX_THREAD_FREELIST_TIMEOUT_2	0x0058		
Reserved	0x005C	RX_THREAD_FREELIST_TIMEOUT_3 in the IXP2400	
NOTES:			
1. Addresses that are used in IXP2400 are reserved in IXP2800 to allow for a common superset address map.			

Table 5-44. MSF Register Summary (Sheet 2 of 5)

Register	Offset (Hex)	Comment	Section
TX_SEQUENCE_0	0x0060		Section 5.7.23
TX_SEQUENCE_1	0x0064		
TX_SEQUENCE_2	0x0068		
Reserved	0x006C	TX_SEQUENCE_3 in the IXP2400	
TX_CALENDAR_LENGTH	0x0070		Section 5.7.26
Reserved	0x0074-0x0088	TX_MPHY_POLL_LIMIT, TX_MPHY_STATUS, TX_MPHY_FORCE_UPDATE, RX_UP_CONTROL, and TX_UP_CONTROL in the IXP2400	
Reserved	0x008C-0x009C		
TRAIN_DATA	0x00A0		Section 5.7.31
TRAIN_CALENDAR	0x00A4		Section 5.7.32
TRAIN_FLOW_CONTROL	0x00A8		Section 5.7.33
Reserved	0x00AC-0x00FC		
FCIFIFO	0x0100-0x013C	FCIFIFO has 16 addresses for burst access. Burst must start at lowest address.	Section 5.7.10
FCEFIFO	0x0140-0x017C	FCEFIFO has 16 addresses for burst access. Burst must start at lowest address.	Section 5.7.11
Reserved	0x0180-0x02FC		
NOTES:			
1. Addresses that are used in IXP2400 are reserved in IXP2800 to allow for a common superset address map.			

Table 5-44. MSF Register Summary (Sheet 3 of 5)

Register	Offset (Hex)	Comment	Section		
RX_DESKEW_RDAT0	0x0300	These registers hold the deskew values per pin from training	Section 5.7.12		
RX_DESKEW_RDAT1	0x0304				
RX_DESKEW_RDAT2	0x0308				
RX_DESKEW_RDAT3	0x030C				
RX_DESKEW_RDAT4	0x0310				
RX_DESKEW_RDAT5	0x0314				
RX_DESKEW_RDAT6	0x0318				
RX_DESKEW_RDAT7	0x031C				
RX_DESKEW_RDAT8	0x0320				
RX_DESKEW_RDAT9	0x0324				
RX_DESKEW_RDAT10	0x0328				
RX_DESKEW_RDAT11	0x032C				
RX_DESKEW_RDAT12	0x0330				
RX_DESKEW_RDAT13	0x0334				
RX_DESKEW_RDAT14	0x0338				
RX_DESKEW_RDAT15	0x033C				
RX_DESKEW_RCTL	0x0340				
RX_DESKEW_RPAR	0x0344				
RX_DESKEW_RPROT	0x0348				
SPI4_DYNFILT_THRESH	0x034C		Section 5.7.13		
MSF_DLL_DATA_DELAY_CTL	0x0350		Section 5.7.14		
RX_DESKEW_RXCSOF	0x0354		Section 5.7.12		
RX_DESKEW_RXCDAT0	0x0358				
RX_DESKEW_RXCDAT1	0x035C				
RX_DESKEW_RXCDAT2	0x0360				
RX_DESKEW_RXCDAT3	0x0364				
RX_DESKEW_RXPAR	0x0368				
RX_DESKEW_RXCSRB	0x036C				
FC_DYNFILT_THRESH	0x0370		Section 5.7.15		
FC_DLL_DATA_DELAY_CTL	0x0374		Section 5.7.16		
Reserved	0x0378–0x037C				
TX_MULTIPLE_PORT_STATUS_#	0x0380–0x03BC		Section 5.7.29		
Reserved	0x03C0–0x03FC				
NOTES:					
1. Addresses that are used in IXP2400 are reserved in IXP2800 to allow for a common superset address map.					

Table 5-44. MSF Register Summary (Sheet 4 of 5)

Register	Offset (Hex)	Comment	Section		
RX_PHASEMON_RDAT0	0x0400	Allows the user to monitor which dll output clock phase is being used to sample the data	Section 5.7.34		
RX_PHASEMON_RDAT1	0x0404				
RX_PHASEMON_RDAT2	0x0408				
RX_PHASEMON_RDAT3	0x040C				
RX_PHASEMON_RDAT4	0x0410				
RX_PHASEMON_RDAT5	0x0414				
RX_PHASEMON_RDAT6	0x0418				
RX_PHASEMON_RDAT7	0x041C				
RX_PHASEMON_RDAT8	0x0420				
RX_PHASEMON_RDAT9	0x0424				
RX_PHASEMON_RDAT10	0x0428				
RX_PHASEMON_RDAT11	0x042C				
RX_PHASEMON_RDAT12	0x0430				
RX_PHASEMON_RDAT13	0x0434				
RX_PHASEMON_RDAT14	0x0438				
RX_PHASEMON_RDAT15	0x043C				
RX_PHASEMON_RCTL	0x0440				
RX_PHASEMON_RPAR	0x0444				
RX_PHASEMON_RPROT	0x0448				
Reserved	0x044C 0x0450				
RX_PHASEMON_RXCSOF	0x0454		Section 5.7.34		
RX_PHASEMON_RXCDAT0	0x0458				
RX_PHASEMON_RXCDAT1	0x045C				
RX_PHASEMON_RXCDAT2	0x0460				
RX_PHASEMON_RXCDAT3	0x0464				
RX_PHASEMON_RXCPAR	0x0468				
RX_PHASEMON_RXSRB	0x046C				
Reserved	0x0470– 0x04FC				
Rx_Port_Calendar_Status_# (0 TO 255)	0x500– 0x8FC		Section 5.7.25		
Reserved	0x0900– 0x0FFC				
TX_CALENDAR_# (0 TO 255)	0x1000– 0x13FC		Section 5.7.27		
TX_PORT_STATUS_# (0 TO 255)	0x1400– 0x17FC		Section 5.7.28		
NOTES:					
1. Addresses that are used in IXP2400 are reserved in IXP2800 to allow for a common superset address map.					

Table 5-44. MSF Register Summary (Sheet 5 of 5)

Register	Offset (Hex)	Comment	Section
TBUF_ELEMENT_CONTROL_# (0 TO 127)	0x1800– 0x1BFC	Write TBUF_ELEMENT_CONTROL_# and set Element Valid. If done as 2 separate 32-bit writes, the write to the upper half of the register sets Element Valid.	Section 5.7.30
RBUF/TBUF (8KB)	0x2000– 0x3FFF	Read = RBUF, Write = TBUF refer to Section 4.1.5 for memory map for the RBUF and TBUF.	Section 4.1.5
MSF_IO_BUF_CTL	0x8008	MSF Rcomp Registers	Section 5.7.35
FC_IO_BUF_CTL	0x800C		Section 5.7.36
NOTES:			
1. Addresses that are used in IXP2400 are reserved in IXP2800 to allow for a common superset address map.			

5.7.1 MSF_RX_CONTROL

The control register defines a number of receive configuration parameters.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RX_EN_C	RX_EN_S	RESERVED		SPI4_CHECKSUM_MODE	DATA_DIP4_DIS	FLWCTL_VPAR_TYPE	DATA_VPAR_TYPE	FLWCTL_VPAR_DIS	DATA_VPAR_DIS	FLWCTL_HPAR_DIS	DATA_HPAR_DIS	RSTAT_OV_VALUE	RSTAT_OVERRIDE	DUPLEX_MODE	RX_CWRD_SIZE	STAT_CLOCK	RSTAT_SELECT	RX_CALENDAR_MODE	RESERVED	CSIX_FREELIST	RESERVED										

Bits	Field	Description	RW	Reset
[31]	RX_EN_C	Receive Enable for CSIX (RPROT = 1). 0—Receive section ignores CSIX transfers on the rx pins. 1—Receive section enabled to receive CSIX transfers as defined in this spec.	RW	0
[30]	RX_EN_S	Receive Enable for SPI-4 (RPROT = 0). 0—Receive section ignores SPI-4 transfers on the rx pins. 1—Receive section enabled to receive SPI-4 transfers as defined in this spec.	RW	0
[29:28]	RESERVED	Reserved	RW	0
[27]	SPI4_CHECKSUM_MODE	Rev A -- Reserved Rev B-- SPI4 checksum mode select 0— 1's compliment of checksum calculated over data words in element (A0 mode) 1— checksum calculated over data words in element	RW	0
[26]	DATA_DIP4_DIS	0—Check DIP-4 Parity received in SPI-4 bursts received on RDAT. 1—Ignore DIP4 field in all SPI4 Control Words received on RDAT. Note - Does not stop the IXP2800 from generating correct DIP-4 Parity on transmitted SPI-4 bursts.	RW	0
[25]	FLWCTL_VPAR_TYPE	Flow Control Vertical parity type. There is an option for CSIX mode to select the type of parity generated and sent inband (with in the CFrame). Refer to bit [24] (DATA_VPAR_TYPE) for more information. 0—Vertical Parity (as defined in CSIX-L1 specification) 1—DIP-16 Parity	RW	0

Bits	Field	Description	RW	Reset
[24]	DATA_VPAR_TYPE	There is an option for CSIX mode to select the type of parity generated and sent inband (within the CFrame). 0—Vertical Parity (as defined in CSIX-L1 specification) 1—DIP-16 Parity (not defined in CSIX Specification. This eliminates the need for the out-of-band horizontal parity specified by CSIX. The DIP-16 parity is generation as defined in SPI-4 specification (which describes a DIP-4 example that produces an intermediate DIP-16).	RW	0
[23]	FLWCTL_VPAR_DIS	0—Calculate the Vertical Parity for the CFrames on the RXCDAT pins and check it with the Vertical Parity received in the CFrame. 1—Ignore the Vertical Parity. Note - Does not stop the IXP2800 from generating correct Vertical Parity on transmitted CFrames.	RW	0
[22]	DATA_VPAR_DIS	0—Calculate the Vertical Parity for the CFrames on the RDAT pins and check it with the Vertical Parity received in the CFrame. 1—Ignore the Vertical Parity. Note - Does not stop the IXP2800 from generating correct Vertical parity on transmitted CFrames.	RW	0
[21]	FLWCTL_HPAR DIS	0—Calculate the Horizontal Parity for the CFrames on the RXCDAT pins and check it with the Horizontal Parity received on the RXCPAR pins. 1—Ignore RXCPAR. Note - Does not stop the IXP2800 from generating correct TXCPAR on transmitted CWords.	RW	0
[20]	DATA_HPAR_DIS	0— Calculate the Horizontal Parity for the CFrames on the RDAT pins and check it with the Horizontal Parity received on the RPAR pins. 1—Ignore RPAR. Note - Does not stop the IXP2800 from generating correct TPAR on transmitted CWords	RW	0
[19:18]	RSTAT_OV_VALUE	Rev A -- Value used with RSTAT_Override. When RSTAT_Override is 0 the value in the calendar is: RBUF above HWM - 10 (Satisfied) RBUF not above HWM - Value in this field When RSTAT_Override is 1 the value in the calendar is: Value in this field Rev B -- This value is used according to the mode specified in RX_CALENDAR_MODE, if RSTAT_OVERRIDE is set.	RW	0

Bits	Field	Description	RW	Reset
[17]	RSTAT_OVERRIDE	<p>Rev A --</p> <p>Allows software to control value sent on RSTAT outputs.</p> <p>0—RSTAT sends calendar based on RBUF High Water Mark and RSTAT_OV_Value as defined in the RSTAT_OV_Value field description (note that it is the more conservative of RBUF_HWM and the RSTAT_OV_Value field.</p> <p>1—RSTAT sends calendar -- status is always equal to RSTAT_OV_Value bits</p> <p>RSTAT calendar is only sent when Train_Data[RSTAT_En] is set, otherwise it is held in framing state.</p> <p>Rev B --</p> <p>Enables the value in RSTAT_OV_VALUE to be sent on the RSTAT outputs according to the mode specified in RX_CALENDAR_MODE.</p>	RW	0
[16]	DUPLEX_MODE	<p>Determines the way CSIX Switch Fabric flow control information is communicated.</p> <ul style="list-style-type: none"> • 0—Simplex Mode. Directly from Switch Fabric. • 1—Full Duplex Mode. From Egress IXP2800 to Ingress IXP2800 <p>Determines the source of the data put into FCEFIFO.</p> <ul style="list-style-type: none"> • Simplex Mode. ME or Intel XScale® core writes to FCEFIFO. • Full Duplex Mode. CSIX CFrames received from Switch Fabric on the RDAT. 	RW	0
[15:14]	RX_CWRD_SIZE	<p>Determines the CWord size on receive (only applies when CFrames are received). The CWord size determines where padding bytes are in a CFrame. (Only applies to RDAT pins; CFrames received into FCIFIFO on RXCDAT always use 32 bit CWords.)</p> <p>00—32 bits 01—64 bits 10—96 bits 11—128 bits</p>	RW	0
[13]	RSTAT_CLOCK	<p>Selects which edge of RSCLK is used to change RSTAT[1:0]. Only applies if RSTAT_Select bit is 0.</p> <p>0—Falling edge 1—Rising edge</p>	RW	0
[12]	RSTAT_SELECT	<p>Selects which pins are used for SPI-4 Status Channel outputs.</p> <p>0—Use LVTTTL pins RSCLK and RSTAT[1:0]. 1—Use LVDS pins TXCCLK and TXCDAT[1:0].</p> <p>LVDS mode must be used when RCLK is greater than or equal to 500 MHz.</p>	RW	0

Bits	Field	Description	RW	Reset
[11]	RX_CALENDAR_MODE	<p>Rev A -- Reserved</p> <p>Rev B -- This bit controls the value driven on RSTAT.</p> <p>0—Conservative_Value The value driven as the status for a given port is the most conservative of:</p> <ul style="list-style-type: none"> the RBUF status based on RBUF high-water-mark (Section 5.7.17). RSTAT_Ov_Value (if RSTAT_Override is 1) (Section 5.7.1) the value in Rx_Port_Calendar_Status_# (Section 5.7.25) for the port. <p>1—Force_Override. The value in Rx_Port_Calendar_Status_# (Section 5.7.25) as described in Rx_Port_Calendar_Status_# description.</p>	RW	0
[10]	RESERVED	Reserved	RW	0
[9]	CSIX_FREELIST	<p>Determines how received CFrames are mapped to RX_THREAD_FREELISTS. See Table 5-45.</p> <p>0—Data and Control CFrames go to different RX_THREAD_FREELISTS.</p> <p>1—Data and Control CFrames go to the same RX_THREAD_FREELIST.</p>	RW	0
[8]	RESERVED	Reserved	RW	0
[7:6]	RBUF_ELE_SIZE_2	<p>Indicates element size for partition 2 of RBUF. Number of elements is a function of number of partitions—see Table 5-45. Only used if RBUF_Partition is 3.</p> <p>00—64 bytes 01—128 bytes 10—256 bytes 11—Reserved</p>	RW	0
[5:4]	RBUF_ELE_SIZE_1	<p>Indicates element size for partition 1 of RBUF. Number of elements is a function of number of partitions—see Table 5-45. Only used if RBUF_Partition is 2 or 3.</p> <p>00—64 bytes 01—128 bytes 10—256 bytes 11—Reserved</p>	RW	0
[3:2]	RBUF_ELE_SIZE_0	<p>Indicates element size for partition 0 of RBUF. Number of elements is a function of number of partitions—see Table 5-45.</p> <p>00—64 bytes 01—128 bytes 10—256 bytes 11—Reserved</p>	RW	0
[1:0]	RBUF_PARTITION	<p>Controls the number of partitions for RBUF elements.</p> <ul style="list-style-type: none"> 00—1 way—All elements allocated from RBUF_Element_Freelist_0 (and RBUF_Element_Freelist_1 based on value in Rx_Port_Map CSR). 01—2 way—$\frac{3}{4}$ of elements allocated from RBUF_Element_Freelist_0 and $\frac{1}{4}$ from RBUF_Element_Freelist_1. 10—3 way—$\frac{1}{2}$ of elements allocated from RBUF_Element_Freelist_0, $\frac{3}{8}$ from RBUF_Element_Freelist_1, and $\frac{1}{8}$ from RBUF_Element_Freelist_2. 11—Reserved 	RW	0

Table 5-45. Number of Elements per RBUF or TBUF Partition

TBUF_PARTITION or RBUF_PARTITION Fields	TBUF_ELE_SIZE_# or RBUF_ELE_SIZE_# Fields	Partition Number		
		0	1	2
00 (1 partition)	00 (64 byte)	128	Unused	Unused
	01 (128 byte)	64		
	10 (256 byte)	32		
01 (2 partitions)	00 (64 byte)	96	32	Unused
	01 (128 byte)	48	16	
	10 (256 byte)	24	8	
10 (3 partitions)	00 (64 byte)	64	48	16
	01 (128 byte)	32	24	8
	10 (256 byte)	16	12	4
This table applies to both RBUF and TBUF partitioning.				

5.7.2 MSF_TX_CONTROL

The control register defines a number of receive configuration parameters.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TX_EN_CD	TX_EN_CC	TX_EN_S	SPARE		TX_FLUSH_PAR2	TX_FLUSH_PAR1	TX_FLUSH_PAR0	SPARE	TX_STATUS_UPDATE_MODE	TX_STATUS_READ_MODE	TXCCLK_SOURCE	FLWCTL_VPAR_TYPE	DATA_VPAR_TYPE	TX_DUPLEX_MODE	TX_CWRD_SIZE	TSTAT_CLOCK	TSTAT_SELECT	TSTAT_SELECT	TCLK_SOURCE	TX_ENABLE	TX_IDLE	TSTAT_EN	TSTAT_EN	TBUF_FILE_SIZE_2	TBUF_FILE_SIZE_1	TBUF_FILE_SIZE_0	TBUF_FILE_SIZE_0	TBUF_FILE_SIZE_0	TBUF_PARTITION		

Bits	Field	Description	RW	Reset
[31]	TX_EN_CD	Transmit Enable for CSIX Data. 0—Do not transmit TBUF elements from CSIX Data partition. The effect is the same as if no CSIX Data TBUF element is valid. 1—Transmit valid CSIX Data TBUF elements as defined in this spec. Whenever TX_EN_CD is cleared, the TX_FLUSH bits of the used partitions must be set to reset the partition <i>before</i> resetting TX_EN_CD to re-enable the partition.	RW	0
[30]	TX_EN_CC	Transmit Enable for CSIX Control. 0—Do not transmit TBUF elements from CSIX Control partition. The effect is the same as if no CSIX Control TBUF element is valid. 1—Transmit valid CSIX Control TBUF elements as defined in this spec. Whenever TX_EN_CC is cleared, the TX_FLUSH bits of the used partitions must be set to reset the partition <i>before</i> resetting TX_EN_CC to re-enable the partition.	RW	0
[29]	TX_EN_S	Transmit Enable for SPI-4. 0—Do not transmit TBUF elements from SPI-4 partition. The effect is the same as if no SPI-4 TBUF element is valid. 1—Transmit valid SPI-4 TBUF elements as defined in this spec. Whenever TX_EN_SS is cleared, the TX_FLUSH bits of the used partitions must be set to reset the partition <i>before</i> resetting TX_EN_SS re-enable the partition.	RW	0
[28:27]	SPARE	Reserved	RW	0
[26]	TX_FLUSH_PAR2	These bits can be written to flush valid entries from TBUF. When a 1 is written to these bits, all valid bits of corresponding TBUF partition elements are cleared, and the internal element pointers and TX sequence number used by that section of TBUF are reset. These bits return 0 when read.	WO	0
[25]	TX_FLUSH_PAR1		WO	0
[24]	TX_FLUSH_PAR0		WO	0
[23]	SPARE	Reserved	RW	0

Bits	Field	Description	RW	Reset
[22]	TX_STATUS_UPDATE MODE	Rev A -- Reserved Rev B -- Controls how TX_PORT_STATUS and TX_MULTIPLE_PORT_STATUS are updated. 0—The latest received status sample for the port is always updated into TX_PORT_STATUS and TX_MULTIPLE_PORT_STATUS. 1—The latest received status sample for the port is updated into TX_PORT_STATUS and TX_MULTIPLE_PORT_STATUS as shown in Table 5-46 .	RW	0
[21]	TX_STATUS_READ MODE	Rev A -- Reserved Rev B --Controls what value is placed into TX_PORT_STATUS and TX_MULTIPLE_PORT_STATUS when they are read. The modified value is used as an indication that the most recently received calendar status has been read. 0—Illegal value (11) 1—Satisfied value (10)	RW	0
[20:19]	TXCCLK_SOURCE	Selects which clock source is used for TXCCLK Output. 00—RCLK input 01—TCLK_REF input 10— Divided version of internal fast clock; divide value is specified in Clock_Control CSR. 11—Reserved	RW	0
[18]	FLWCTL_VPAR_TYPE	Flow Control Vertical Parity Type 0—Vertical Parity (defined in CSIX-L1 specification) 1—DIP-16 Parity (defined in SPI-4 specification)	RW	0
[17]	DATA_VPAR_TYPE	DATA Vertical Parity Type 0—Vertical Parity (defined in CSIX-L1 specification) 1—DIP-16 Parity (defined in SPI-4 specification) This bit is available for CSIX mode only.	RW	0
[16]	TX_DUPLEX_MODE	Determines the way CSIX Switch Fabric flow control information is communicated. 0—Simplex Mode. Directly from Switch Fabric. 1—Full Duplex Mode. From Egress IXP2800 to Ingress IXP2800 Determines where CSIX link level Flow Control bits saved in the FC_Ingress_Status CSR are derived. Simplex Mode. Directly from Switch Fabric via the RXCDAT, RXCPAR, RXCSOF pins. Full Duplex Mode. From the Egress processor via the RXCSRB pin.	RW	0
[15:14]	TX_CWRD SIZE	Determines the CWord size on transmit (only applies when CFrames are transmitted). The CWord size determines where padding bytes are in a CFrame. (Only applies to TDAT pins; CFrames transmitted from FCEFIFO on TXCDAT always use 32 bit CWords.) 00—32 bits 01—64 bits 10—96 bits 11—128 bits	RW	0
[13]	TSTAT_CLOCK	Selects which edge of TCLK is used to sample TSTAT[1:0]. Only applies if TSTAT_Select bit is 0. 0—Falling edge 1—Rising edge	RW	0

Bits	Field	Description	RW	Reset
[12]	TSTAT_SELECT	Selects which pins are used for SPI-4 Status Channel inputs. 0—Use LVTTTL pins TSCLK and TSTAT[1:0]. 1—Use LVDS pins RXCCLK and RXCDAT[1:0].	RW	0
[11]	TCLK_SOURCE	Selects the source for TCLK output. 0—From TCLK_REF input. 1—Divided version of internal fast clock; divide value is specified in CLOCK_CONTROL CSR.	RW	0
[10]	TX_ENABLE	Enables transmission. 0—All Transmit output signals remain driven low. 1—Normal transmit operation is enabled. Note this bit does not control TCLK. TCLK is controlled by MSF_CLK_CONTROL[TCLK_EN].	RW	0
[9]	TX_IDLE	Controls which type of idle information is transmitted if no TBUF element is valid or if transmit flow control has disabled transmission. 0—SPI-4 1—CSIX Note: in interleaved mode, where the TBUF is partitioned to transmit CSIX and SPI-4 data, this bit must be set to 1.	RW	0
[8]	TSTAT_EN	TSTAT Enable. 0—TSTAT pins are ignored. 1—TSTAT is used to receive FIFO status according to the description in SPI-4 Transmit Flow Control. Note: enables writing RXCDAT in LVDS mode.	RW	0
[7:6]	TBUF_ELE_SIZE_2	Indicates element size for partition 2 of TBUF. Number of elements is a function of number of partitions—see Table 5-45. Only used if TBUF_Partition is 3. 00—64 bytes 01—128 bytes 10—256 bytes 11—Reserved	RW	0
[5:4]	TBUF_ELE_SIZE_1	Indicates element size for partition 1 of TBUF. Number of elements is a function of number of partitions—see Table 5-45. Only used if TBUF_Partition is 2 or 3. 00—64 bytes 01—128 bytes 10—256 bytes 11—Reserved	RW	0
[3:2]	TBUF_ELE_SIZE_0	Indicates element size for partition 0 of TBUF. Number of elements is a function of number of partitions—see Table 5-45. 00—64 bytes 01—128 bytes 10—256 bytes 11—Reserved	RW	0
[1:0]	TBUF_PARTITION	Controls the number of partitions for TBUF elements. 00—1 way—All elements allocated from TBUF_Element_Freelist_0. 01—2 way— $\frac{3}{4}$ of elements allocated from TBUF_Element_Freelist_0 and $\frac{1}{4}$ from TBUF_Element_Freelist_1. 10—3 way— $\frac{1}{2}$ of elements allocated from TBUF_Element_Freelist_0, $\frac{3}{8}$ from TBUF_Element_Freelist_1, and $\frac{1}{8}$ from TBUF_Element_Freelist_2. 11—Reserved	RW	0

Table 5-46. New Port Status to be saved based on currently saved value and new value received on TSTAT

	TX_STATUS_UPDATE_MODE Value				
	1				0
New Value from TSTAT	Starving	Hungry	Satisfied	Illegal	Any
Starving	Starving	Starving	Starving	Starving	Starving
Hungry	Starving	Hungry	Hungry	Hungry	Hungry
Satisfied	Starving	Hungry	Satisfied	Satisfied	Satisfied

5.7.3 MSF_INTERRUPT_STATUS

This register holds error status. When any of these bits is set, and the corresponding bit in MSF_INTERRUPT_ENABLE is set, MSF generates an interrupt signal to the Intel XScale® processor. Note that these bits can be read even if the interrupt is not enabled.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
FCEFIFO_OVFLW_CNT								RBUF_OVFLW_CNT										FCIFIFO_PARITY_ERR	FLWCTL_TRAIN_STOPPED	CAL_TRAIN_STOPPED	DATA_TRAIN_STOPPED	DET_CSIX_FC_IDLE	DET_CSIX_IDLE	REC_FLW_CTL_TRAIN	REC_CAL_TRAIN	REC_DATA_TRAIN	FCIFIFO_ERR	DETECT_NO_CAL	TBUF_ERROR	TSTAT_PAR_ERR	DIP4_ERR	VP_ERR	HP_ERR

Bits	Field	Description	RW	Reset
[31:24]	FCEFIFO_OVFLW_CNT	<p>Full Duplex Mode: CSIX CFrame mapped to FCEFIFO arrived and FCEFIFO did not have sufficient room. The entire CFrame is discarded. This field counts up by 1 for every discarded CFrame.</p> <p>Simplex Mode: (IXP2800 Rev B ONLY) Software attempted to write data into FCEFIFO and it is completely full. (FCEFIFO can hold up to 255 entries in this mode). This field counts up by 1 for every discarded CWord.</p> <p>This field saturates at 0xFF. In other words, when the count reaches 0xFF, it will stop counting, so as to not roll over to zero.</p> <p>Note that these bits need to be cleared all at the same time by writing 0xFF to them.</p>	RW 1C	0
[23:16]	RBUF_OVFLW_CNT	<p>Data was received on Rx pins and no buffer space was free to accept it. The data is discarded. This field counts up by 1 for every discarded SPI-4 burst or CFrame. If the count reaches 0xFF it will stop counting, so as to not roll over to zero.</p> <p>Note that these bits need to be cleared all at the same time by writing 0xFF to them.</p>	RW 1C	0
[15]	FCIFIFO_PARITY_ERR	Incorrect Parity was received on a CFrame received on RXCDAT pins.	RW 1C	0
[14]	FLWCTL_TRAIN_STOPPED	When set indicates that Training was received on Flow Control Rx pins and the training sequence has stopped.	RW 1C	0
[13]	CAL_TRAIN_STOPPED	When set indicates that Training was received on Calendar Rx pins and the training sequence has stopped.	RW 1C	0
[12]	DATA_TRAIN_STOPPED	When set indicates that Training was received on Data Rx pins and the training sequence has stopped.	RW 1C	0
[11]	DET_CSIX_FC_IDLE	When set indicates that at least one CSIX Idle CFrame was received by the IXP2800 on the RXCDAT pins.	RW 1C	0
[10]	DET_CSIX_IDLE	When set indicates that at least one CSIX Idle CFrame was received by the IXP2800 on the RDATA pins.	RW 1C	0
[9]	REC_FLW_CTL_TRAIN	At least one complete training sequence was received on the Flow Control Rx pins. If this bit is cleared by software while continuous training is being received, it will set again.	RW 1C	0
[8]	REC_CAL_TRAIN	At least one complete training sequence was received on the Calendar Rx pins. If this bit is cleared by software while continuous training is being received, it will set again.	RW 1C	0
[7]	REC_DATA_TRAIN	At least one complete training sequence was received on the Data Rx pins. If this bit is cleared by software while continuous training is being received, it will set again.	RW 1C	0

Bits	Field	Description	RW	Reset
[6]	FCIFIFO_ERR	FCIFIFO CFrame was discarded due to either: Horizontal Parity Error Vertical Parity Error Premature RXCSOF (before entire payload length was received) Overflow—CFrame with Payload Size greater than space available in FCIFIFO	RW 1C	0
[5]	DETECT_NO_CAL	Status channel disabled indicator. Status channel is either TSTAT or RXCDAT depending on MSF_Tx_Control[TSTAT_Select]. When LVTTTL is used, this bit indicates TSTAT input has had framing pattern for more than 32 consecutive cycles. When LVDS is used, this bit indicates that 3 consecutive training patterns were detected on RXCDAT. <ul style="list-style-type: none"> 0—Status channel indicates calendar is active. 1—Status channel indicates need for training (i.e. no calendar active). 	RW 1C	0
[4]	TBUF_ERROR	Transmit Control Word programming error. When set indicates that a programming error occurred when writing the TBUF_Element_Control register. <ul style="list-style-type: none"> In CSIX and SPI4 modes, this bit is set when the sum of ((prepend_offset + prepend_length + 7) & 0xf8) + payload_offset + payload_length is greater than the TBUF element size. In this case the payload length transmitted will be truncated by the number of bytes in excess of the TBUF element size. In SPI4 mode, this bit is also set when the sum of the Prepend Length and Payload Length is not an integer multiple of 16 bytes and the EOP bit is not set in the Transmit Control Word. 	RW 1C	0
[3]	TSTAT_PAR_ERR	Incorrect DIP-2 Parity was received on TSTAT or RXCDAT.	RW 1C	0
[2]	DIP4_ERR	Incorrect DIP-4 Parity on received SPI-4.	RW 1C	0
[1]	VP_ERR	Incorrect Vertical Parity on a received CFrame or premature SOF.	RW 1C	0
[0]	HP_ERR	Incorrect Horizontal Parity on a received CFrame.	RW 1C	0

5.7.4 MSF_INTERRUPT_ENABLE

This register holds enable bits for individual error types. This register is bitwise ANDed with MSF_INTERRUPT_STATUS—if the result is not zero MSF generates an interrupt signal to the Intel XScale® processor

3	3	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0														
RESERVED												FCEFIFO_OVFLW_CNT	RESERVED												RBUF_OVFLW_CNT	FCIFIFO_PARITY_ERR	FLWCTL_TRAIN_STOPPED	CAL_TRAIN_STOPPED	DATA_TRAIN_STOPPED	DET_CSIX_FC_IDLE	DET_CSIX_IDLE	REC_FLW_CTL_TRAIN	REC_CAL_TRAIN	REC_TRAIN	FCIFIFO_ERR	DETECT_NO_CAL	TBUF_ERROR	TSTAT_PAR_ERR	DIP4_ERR	VP_ERR	HP_ERR

Bits	Field	Description	RW	Reset
[31:25]	RESERVED	Reserved	RO	0
[24]	FCEFIFO_OVFLW_CNT	This bit is ANDed with the OR of all the FCEFIFO_OVERFLOW_COUNT bits (to detect when the count is not zero).	RW	0
[23:17]	RESERVED	Reserved	RO	0
[16]	RBUF_OVFLW_CNT	This bit is ANDed with the OR of all the RBUF_OVERFLOW_COUNT bits (to detect when the count is not zero).	RW	0
[15]	FCIFIFO_PARITY_ERR	Incorrect Parity was received on a CFrame received on RXCDAT pins.	RW	0
[14]	FLWCTL_TRAIN_STOPPED		RW	0
[13]	CAL_TRAIN_STOPPED		RW	0
[12]	DATA_TRAIN_STOPPED		RW	0
[11]	DET_CSIX_FC_IDLE		RW	0
[10]	DET_CSIX_IDLE		RW	0
[9]	REC_FLW_CTL_TRAIN		RW	0
[8]	REC_CAL_TRAIN		RW	0
[7]	REC_TRAIN		RW	0
[6]	FCIFIFO_ERR		RW	0
[5]	DETECT_NO_CAL		RW	0
[4]	TBUF_ERROR		RW	0
[3]	TSTAT_PAR_ERR		RW	0
[2]	DIP4_ERR		RW	0
[1]	VP_ERR		RW	0
[0]	HP_ERR		RW	0

5.7.5 CSIX_TYPE_MAP

CSIX supports a 4-bit Type field (16 types) in the base header. When a CFrame is received, the Type field is checked and the CFrame is directed based on the settings in this register. This register must not be changed while CSIX CFrames are being received.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0		
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
TYP 15		TYP 14		TYP 13		TYP 12		TYP 11		TYP 10		TYP 9		TYP 8		TYP 7		TYP 6		TYP 5		TYP 4		TYP 3		TYP 2		TYP 1		TYP 0	

Bits	Field	Description	RW	Reset
[31:0]	TYP15 -TYP0	<p>These pairs of bits map CFRAME TYPE field as shown. Bit [1:0] is for type 0x0, bit[3:2] for type 0x1, etc.</p> <p>00—Discard 01—RBUF Control partition 10—RBUF Data partition 11—Flow Control Egress FIFO (FCEFIFO)</p> <p>CFrames type defined in Version 1 of the CSIX-L1 are:</p> <p>0x0 always discarded 0x6 Flow Control Frame 0x1 Unicast 0x7 Command and Status 0x2 Multicast Mask 0x8-0xB CSIX Reserved 0x3 Multicast ID 0xC-0xF Private 0x4 Multicast Binary Copy 0x5 Broadcast</p>	RW	0

5.7.6 FC_EGRESS_STATUS

This register holds the link level flow control information received from the Switch Fabric, and the status of RBUF and FCEFIFO. In Full Duplex Mode, this information is transmitted out of the IXP2800 on TXCSRB.

Note: The actual update into bits[1:0] of this register is done so as to be pessimistic. When a Base Header is received with Ready bit(s) deasserted, 0 is put into the appropriate bit(s) prior to receiving the entire CFrame. When a Base Header is received with Ready bit asserted, the 1 is held until the entire CFrame is received error-free (i.e. wait until the Vertical Parity field is found to be good) before reflecting that value in the appropriate bit(s). If there is an error on a CFrame, or if the Switch Fabric is transmitting Training Sequence or continuous Dead Cycles, then those two bits are cleared (regardless of the Ready bits received). CFrame errors will set bits as defined in MSF_Interrupt_Status to assist in debug.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0									
RESERVED																								FCEFIFO_FULL	TM_DREADY	TM_CREADY	SF_DREADY	SF_CREADY		

Bits	Field	Description	RW	Reset
[31:5]	RESERVED	Reserved	RO	0
[4]	FCEFIFO_FULL	Indicates if FCEFIFO is full or not, based on HWM_CONTROL[FCEFIFO_HWM]. Note that this bit is primarily used in Simplex Mode.	RO	0
[3]	TM_DREADY	Value for Ingress IXP2800 to use for Data Ready in CFrames sent to Switch Fabric. When FC_STATUS_OVERRIDE[EGRESS_FORCE_EN] is 0 -- Deasserted when RBUF CSIX Data Partition is full (based on HWM_Control[RBUF_D_HWM]). When FC_STATUS_OVERRIDE[EGRESS_FORCE_EN] is 1 – The value in FC_STATUS_OVERRIDE[4] In Full Duplex Mode—transmitted on TXCSRB bit 9. In Simplex mode—Data Ready value to transmit (in bit 7 of byte 0) of CSIX Base Headers sent on TXCDAT.	RO	0
[2]	TM_CREADY	Value for Ingress IXP2800 to use for Control Ready in CFrames sent to Switch Fabric. When FC_STATUS_OVERRIDE[EGRESS_FORCE_EN] is 0 -- In Full Duplex Mode, deasserted when RBUF CSIX Control Partition is full (based on HWM_Control[RBUF_C_HWM]) or when FCEFIFO is full (based on HWM_Control[FCEFIFO_HWM]). In Simplex Mode deasserted only when RBUF CSIX Control Partition is full. When FC_STATUS_OVERRIDE[EGRESS_FORCE_EN] is 1 – The value in FC_STATUS_OVERRIDE[5]. In Full Duplex Mode—transmitted on TXCSRB bit 8. In Simplex mode—Control Ready value to transmit (in bit 6 of byte 0) of CSIX Base Headers sent on TXCDAT.	RO	0
[1]	SF_DREADY	Data Ready received in CSIX Base Headers (bit 7 of byte 0), updated after the CFrame is received. It is cleared if an error is detected on a received CFrame, or if the Switch Fabric is transmitting Training Sequence or continuous Dead Cycles. When FC_STATUS_OVERRIDE[EGRESS_FORCE_EN] is 0 -- Equals bit in CSIX Base Headers (bit 7 of byte 0) 1 – Equals the value in FC_STATUS_OVERRIDE[6] In Full Duplex Mode—transmitted on TXCSRB bit 7.	RO	0
[0]	SF_CREADY	Control Ready received in CSIX Base Headers (bit 6 of byte 0), updated after CFrame is received. Cleared if an error is detected on a received CFrame, or if the Switch Fabric is transmitting Training Sequence, or continuous Dead Cycles. When FC_STATUS_OVERRIDE[EGRESS_FORCE_EN] is 0 -- Equals bit in CSIX Base Headers (bit 6 of byte 0) 1 – Equals the value in FC_STATUS_OVERRIDE[7] In Full Duplex Mode—transmitted on TXCSRB bit 6.	RO	0

5.7.7 FC_INGRESS_STATUS

This register holds the link level flow control information received on RXCSR. It is used as ready information in CSIX Base Headers, and used to enable or disable transmission of CFrames to the Switch Fabric

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																											TM_DREADY	TM_CREADY	SF_DREADY	SF_CREADY

Bits	Field	Description	RW	Reset
[31:4]	RESERVED	Reserved	RO	0
[3]	TM_DREADY	Data Ready value to transmit in bit 7 of byte 0 of CSIX Base Headers on TDAT pins. When FC_STATUS_OVERRIDE[INGRESS_FORCE_EN] is 0 –values defined below for Full Duplex and Simplex modes respectively. When FC_STATUS_OVERRIDE[INGRESS_FORCE_EN] is 1 – The value in FC_STATUS_OVERRIDE[0] In Full Duplex Mode—received on RXCSR bit 9. Cleared if Training Sequence detected on Flow Control Bus. In Simplex mode—0.	RO	0
[2]	TM_CREADY	Control Ready value to transmit (in bit 6 of byte 0) of CSIX Base Headers on TDAT pins. When FC_STATUS_OVERRIDE[INGRESS_FORCE_EN] is 0 –values defined below for Full Duplex and Simplex modes respectively. When FC_STATUS_OVERRIDE[INGRESS_FORCE_EN] is 1 – The value in FC_STATUS_OVERRIDE[1] In Full Duplex Mode—received on RXCSR bit 8. Cleared if Training Sequence detected on Flow Control Bus. In Simplex mode—Number of free CWords in FCIFIFO is below high water mark as programmed in HWM_Control[FCIFIFO_Ext_HWM] (Note -- CReady is true when FCIFIFO is not nearly full).	RO	0

Bits	Field	Description	RW	Reset
[1]	SF_DREADY	<p>Hardware control of transmission of CSIX Data elements.</p> <ul style="list-style-type: none"> 0—Stop sending Data CFrames to the Switch Fabric. 1—OK to send Data CFrames to the Switch Fabric. <p>When FC_STATUS_OVERRIDE[INGRESS_FORCE_EN] is 0 –values defined below for Full Duplex and Simplex modes respectively.</p> <p>When FC_STATUS_OVERRIDE[INGRESS_FORCE_EN] is 1 – The value in FC_STATUS_OVERRIDE[2]</p> <p>In Full Duplex Mode—received on RXCSR bit 7. Cleared if Training Sequence detected on Flow Control Bus.</p> <p>In Simplex mode—Data Ready of CSIX Base Headers (bit 7 of byte 0) received on RXCDAT, updated as each CFrame is received. Cleared if an error is detected on a received CFrame, or if the Switch Fabric Flow Control Bus is transmitting Training Sequence, or continuous Dead Cycles.</p>	RO	0
[0]	SF_CREASY	<p>Hardware control of transmission of CSIX Control elements.</p> <ul style="list-style-type: none"> 0—Stop sending Control CFrames to the Switch Fabric. 1—OK to send Control CFrames to the Switch Fabric. <p>When FC_STATUS_OVERRIDE[INGRESS_FORCE_EN] is 0, the value that is sent is defined below for Full Duplex and Simplex modes respectively.</p> <p>When FC_STATUS_OVERRIDE[INGRESS_FORCE_EN] is 1, the value in FC_STATUS_OVERRIDE[3] is sent.</p> <p>In Full Duplex Mode, the value sent is the value received on RXCSR bit 6 and is cleared if Training Sequence detected on Flow Control Bus.</p> <p>In Simplex mode, the value sent is the Ready flag for the control traffic which is found in the CSIX Base Headers (bit 6 of byte 0) received on RXCDAT. It is updated as each CFrame is received and cleared if an error is detected on a received CFrame, if the Switch Fabric Flow Control Bus is transmitting Training Sequence, or there are continuous Dead Cycles.</p>	RO	0

5.7.8 FC_STATUS_OVERRIDE

This register sets how the CSIX Ready bits are driven onto the TXSRB and/or RXSRB pins.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																				EGRESS_FORCE_EN	INGRESS_FORCE_EN	EGRESS_FORCE				INGRESS_FORCE					

Bits	Field	Description	RW	Reset
[31:10]	RESERVED	Reserved	RO	0
[9]	EGRESS_FORCE_EN	This bit controls the data driven on TXCSR output. 0 = Drive TXCSR output from FC_EGRESS_STATUS 1 = Drive TXCSR output from EGRESS_FORCE FIELD	RW	1
[8]	INGRESS_FORCE_EN	This bit controls whether RXCSR input or INGRESS_FORCE field is used. 0 = Use RXCSR input signal 1 = Use INGRESS_FORCE field	RW	1
[7:4]	EGRESS_FORCE	This is used to provide hardware with software-chosen values to send on the TXCSR pin in place of the ones generated by internal hardware. 7: SF_CReady 6: SF_DReady 5: TM_CReady 4: TM_DReady	RW	0
[3:0]	INGRESS_FORCE	This is used to provide hardware with software-chosen values to use in place of the ones received on the RXCSR pin. 3: SF_CReady 2: SF_DReady 1: TM_CReady 0: TM_DReady	RW	0

5.7.9 MSF_CLOCK_CONTROL

This register is used to control MSF clocks to allow software to initialize MSF and external Media and Switch Fabric devices. Before this register can be written, the MSF Unit must be taken out of reset by writing a 0 to IXP_RESET_0[MEDIA].

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	0	0																				
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0																						
RESERVED																						FC_RCV_CLK_SEL	RCV_CLK_SEL	RXCCLK_DLL_EN	RCLK_DLL_EN	RSX_SECTION_EN	TSX_SECTION_EN	RCLK_REF_EN	TXCCLK_EN	RSCLK_EN	TCLK_EN	RX_FC_SECTION_EN	TX_FC_SECTION_EN	RX_SECTION_EN	TX_SECTION_EN								

Bits	Field	Description	RW	Reset
[31:14]	RESERVED	Reserved	RO	0x0
[13]	IXP2800 Rev A: FC_RCV_CLK_SEL IXP2800 Rev B: RXCCLK_DLL_RESET	IXP2800 Rev A: 0 – Flow control receive logic uses clock which is divided from main PLL output frequency using divide value in CLOCK_CONTROL[MSF_CLK_RATIO]. 1 – Flow control receive logic uses RXCCLK_DLL output (which is locked to RXCCLK input). This bit must not be written to a 1 until RXCCLK_DLL_EN has been written to a 1 and DLL has been given enough time to lock. IXP2800 Rev B: 0 – Hold the FC DLL in reset. 1 – Remove the FC DLL from reset. This bit must be written to a 1 prior to RXCCLK_DLL_EN is asserted	RW	0x0
[12]	IXP2800 Rev A: RCV_CLK_SEL IXP2800 Rev B: RCLK_DLL_RESET	IXP2800 Rev A: 0 – Data receive logic uses clock which is divided from main PLL output frequency using divide value in CLOCK_CONTROL[MSF_CLK_RATIO]. 1 – Data receive logic uses RCLK_DLL output (which is locked to RCLK input). This bit must not be written to a 1 until RCLK_DLL_EN has been written to a 1 and DLL has been given enough time to lock. IXP2800 Rev B: 0 – Hold the MSF DLL in reset. 1 – Remove the MSF DLL from reset. This bit must be written to a 1 prior to RCLK_DLL_EN is asserted	RW	0x0

Bits	Field	Description	RW	Reset
[11]	RXCCLK_DLL_EN	<p>IXP2800 Rev A: 0 – RXCCLK DLL is held in reset 1 – RXCCLK DLL locks to frequency on RXCCLK input pin. The output of this DLL can be used as source for flow control receive logic.</p> <p>IXP2800 Rev B: 0 - RXCCLK (Flowcontrol) internal clock network is driven by the internal clock pclk. 1 - The DLL output is switched to drive the RXCCLK (Flowcontrol) internal clock network.</p> <p>This bit should not be set until bit 13 (RXCCLK_DLL_RESET) has been set and until the DLL has locked to the input clock.</p>	RW	0x0
[10]	RCLK_DLL_EN	<p>IXP2800 Rev A 0 – RCLK DLL is held in reset. 1 – RCLK DLL locks to frequency on RCLK input pin. The output of this DLL can be selected as the source for RCLK_REF and data receive logic based on RCV_CLK_SEL of this register.</p> <p>IXP2800 Rev B: 0 - RCLK (MSF) internal clock network is driven by the internal clock pclk. 1 - The DLL output is switched to drive the RCLK (MSF) internal clock network.</p> <p>This bit should not be set until bit 13 (RXCLK_DLL_RESET) has been set and until the DLL has locked to the input clock.</p>	RW	0x0
[9]	RSX_SECTION_EN	<p>0 – Receive calendar section of MSF (all functions clocked by RSCLK) are in reset state. 1 – Receive calendar section of MSF runs normally This bit is used to prevent clock glitches from affecting Receive calendar section when RSCLK is being changed. Do not write this bit to a 1 until the RCLK DLL is locked.</p>	RW	0x0
[8]	TSX_SECTION_EN	<p>0 – Transmit calendar section of MSF (all functions clocked by TSCLK) are in reset state. 1 – Transmit calendar section of MSF runs normally This bit is used to prevent clock glitches from affecting Transmit section when TSCLK is being changed.</p>	RW	0x0
[7]	RCLK_REF_EN	<p>0 – RCLK_REF output is statically 0 1 – RCLK_REF output runs at frequency of RCLK RCLK_REF will not have a glitch when this bit is written from 0 to 1.</p>	RW	0x0
[6]	TXCCLK_EN	<p>0 – TXCCLK output is statically 0 1 – TXCCLK output runs at frequency selected by MSF_TX_CONTROL[TXCCLK_SOURCE] TXCCLK will not have a glitch when this bit is written from 0 to 1.</p>	RW	0x0
[5]	RSCLK_EN	<p>0 – RSCLK output is statically 0 1 – RSCLK output runs at ¼ the frequency of RCLK input. RSCLK will not have a glitch when this bit is written from 0 to 1.</p>	RW	0x0



Bits	Field	Description	RW	Reset
[4]	TCLK_EN	0 – TCLK output is statically 0 1 – TCLK output runs at frequency selected by MSF_TX_CONTROL[TCLK_SOURCE] TCLK will not have a glitch when this bit is written from 0 to 1.	RW	0x0
[3]	RX_FC_SECTION_EN	0 – Receive flow control section of MSF (all functions clocked by RXCCLK) are in reset state. 1 – Receive flow control section of MSF runs normally This bit is used to prevent clock glitches from affecting Receive flow control section when RXCCLK is being changed. Do not write this bit to a 1 until the RXCCLK DLL is locked.	RW	0x0
[2]	TX_FC_SECTION_EN	0 – Transmit flow control section of MSF (all functions clocked by TXCCLK) are in reset state. 1 – Transmit flow control section of MSF runs normally This bit is used to prevent clock glitches from affecting Transmit flow control section when TXCCLK is being changed (for example when MSF_TX_CONTROL[TXCCLK_SOURCE] or CLOCK_CONTROL[MSF_CLK_RATIO] is being changed).	RW	0x0
[1]	RX_SECTION_EN	0 – Receive section of MSF (all functions clocked by RCLK) are in reset state. 1 – Receive section of MSF runs normally This bit is used to prevent clock glitches from affecting Receive section when RCLK is being changed. Do not write this bit to a 1 until the RCLK DLL is locked.	RW	0x0
[0]	TX_SECTION_EN	0 – Transmit section of MSF (all functions clocked by TCLK) are in reset state. 1 – Transmit section of MSF runs normally This bit is used to prevent clock glitches from affecting Transmit section when TCLK is being changed (for example when MSF_TX_CONTROL[TCLK_SOURCE] or CLOCK_CONTROL[MSF_CLK_RATIO] is being changed).	RW	0x0

5.7.10 FCIFIFO

This register is used by Microengines to read CFrames, one CWord at a time, from the FCIFIFO. When valid CWords are read they are removed from FCIFIFO. If FCIFIFO is empty when it is read, it substitutes an Idle CWord for the read data.

When reading this register, the number of words returned always equals the value of ref_cnt in the instruction.

In the IXP2800 the vertical parity is written into the FCIFIFO. When reading a CFrame from this register, the programmer must count for reading the vertical parity and the padding. For example, reading a 5 byte data (d) CFrame with two bytes of header (h), 4 bytes of extension header (e) and 2 bytes of vertical parity (v), will involve reading four CWords with the following format: hhhh eeee, eeee dddd, dddd dd00, 0000 vvvv, where the comma separates the CWords read from this register, v is a place holder for the vertical parity, and 0 is padding.

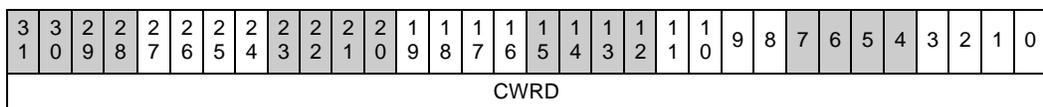
3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
CWRD																															

Bits	Field	Description	RW	Reset
[31:0]	CWRD	Data from head entry of FCIFIFO, or Idle CWord if FCIFIFO is empty.	RO	undef

5.7.11 FCEFIFO

This register is used by Microengines to write CFrames, one CWord at a time, to the FCEFIFO, when MSF_RX_CONTROL[DUPLX_MODE] is Simplex Mode. It is up to Microengines to test for room in FCEFIFO by reading FC_EGRESS_STATUS[FCEFIFO_FULL].

In the IXP2800, the trailing bytes of the last CWord must be set to zero by the software. If the vertical parity does not fit in the last CWord written into the FIFO, an extra CWord equal to zero must be written as a place holder for the vertical parity. For example, writing a 5 byte data (d) CFrame with two bytes of header (h), 4 bytes of extension header (e) and 2 bytes of vertical parity (v) should have the following format: hhhh eeee, eeee dddd, dddd dd00, 0000 vvvv, where the comma separates the CWords written to this register, v is a place holder for the vertical parity, and 0 is padding.



Bits	Field	Description	RW	Reset
[31:0]	CWRD	Data written to tail entry of FCEFIFO.	W	undef

5.7.12 RX_DESKEW_# (# = pin name)

These registers hold the deskew values per pin from training. Normally their use is to read the values to test the training logic. However, if training is disabled in Train_Calendar[IGN_TRAIN], they can be written. MEs cannot use the msf[fast_wr] instruction to write this register.

There is a register per pin; the # in Rx_DeskeW_# is replaced by each pin name. Refer to Table 5-47 for the actual names and addresses.

Table 5-47. List of RX_DESKEW_# Registers

RX_DESKEW_# - Replace # With the Following			
RDAT0	RDAT7	RDAT14	RXC DAT1
RDAT1	RDAT8	RDAT15	RXC DAT2
RDAT2	RDAT9	RCTL	RXC DAT3
RDAT3	RDAT10	RPAR	RXC PAR
RDAT4	RDAT11	RPROT	RXC SRB
RDAT5	RDAT12	RXC SOF	
RDAT6	RDAT13	RXC DAT0	

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																										STATUS	DESKEW_VALUE				

Bits	Field	Description	RW	Reset
[31:4]	RESERVED	Reserved	RO	0
[4]	STATUS	A value of 0 indicates the success and a value of 1 indicates the failure of the most recent training for that pin. A failure means that the dynamic deskew logic was not able to find a clock to use, and the value in the Deskew Value field is not valid. One typical reason a value could not be found is that there was no transition on the signal.	RO	0
[3:0]	DESKEW_VALUE	<p>This field enables software to manually select one of the internally generated deskew clocks to be used. The values are the same as specified in the “RX_PHASEMON_# (# = pin name)” registers.</p> <p>Value - Degree dll clock phase</p> <p>0x0 - 90 degree</p> <p>0x1 - 112.5 degree</p> <p>0x2 - 135 degree</p> <p>0x3 - 157.5 degree</p> <p>0x4 - 180 degree</p> <p>0x5 - 202.5 degree</p> <p>0x6 - 225.0 degree</p> <p>0x7 - 247.5 degree</p> <p>0x8 - 270 degree</p> <p>0x9 - 292.5 degree</p> <p>0xA - 315 degree</p> <p>0xB - 337.5 degree</p> <p>0xC - 0 degree</p> <p>0xD - 22.5degree</p> <p>0xE - 45 degree</p> <p>0xF - 67.5 degree</p> <p>Reading this register returns the value that was last written to this register. The “RX_PHASEMON_# (# = pin name)” registers should be read to get the Degree dll clock phase currently used.</p> <p>This field can be modified only when the RX_SECTION_EN bit and the RX_FC_SECTION_EN bit in the MSF_CLOCK CONTROL register are disabled</p>	RW	0

5.7.13 SPI4_DYNFILT_THRESH

The SPI4_DYNFILT_THRESH and FC_DYNFILT_THRESH registers are enabled when the DYN_DESKEW_DIS bit in the MSF_IO_BUF_CTL and FC_IO_BUF_CTL registers are set. This is the dynamic jitter compensation mode. The feature works as follows. A dll in the receive logic generates 16 sampling clocks for each RCLK cycle. These clocks are referred to as the phase clocks. The phase of each phase clock is skewed 22.5 degrees from the last. Since LVDS data is double data rate, eight of the phase clocks are used to sample each of the two data bits transmitted in each clock cycle. Normally, dynamic deskew mode is used to find the center of each data bit and then logic selects the phase clock that is closest to the center of this data. Dynamic jitter

compensation mode attempts to correct for phase shifts between the clock and data which might occur dynamically and randomly over time after the initial training has been done. Three successive samples are compared to each other every bit time. When these samples are all equal, the phase clock being used to sample incoming data is reasonably centered in the data valid window and no adjustments are made. Occasionally, noise may cause the data and clock to skew from one another by a different amount than was present when training first took place. It is expected that a large percentage of this additional skew will occur over a period of time measured in the 1's to 10's of bit times, caused for example by a change in the supply voltage of the driver. If this skew is big enough, the three samples will no longer be equal. This signifies that the selected phase clock should be adjusted to re-center the sample point. When a mismatch in the three samples is detected, the dynamic jitter logic will adjust the selected phase clock to move to the next or the previous sample depending on whether the mismatched samples indicate the sampling is occurring too early or too late. The DYNFILT_THRESH value allows the user to control the bandwidth of these updates. When a value of 1 is programmed, one mismatched sample will cause the phase adjustment to be made. When a 2 is programmed, the jitter compensation will wait to see 2 mismatched samples (out of the last 4 data transitions) before it makes an adjustment, etc). When values of 2 or higher are set, a delay adjustment sample cancels an advance adjustment sample.

Of special interest regarding the rate of update, it takes the jitter compensation logic 5 bit-times to complete an adjustment operation from the time the sample first arrived. This makes the cut-off frequency of the jitter comp logic 100Mhz when the port is running at 500Mhz. Aliasing could be a problem if the rate of change in the skew between clock and data is too high.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																												#_SAMPLES				

Bits	Field	Description	RW	Reset
[31:3]	RESERVED	Reserved	RO	0x0
[2:0]	#_SAMPLES	The number of mismatched samples that must be seen before an adjustment is made to advance or delay the sampling clock phase. Legal values are 1-4, and 7. A value 7 disables dynamic filtering. This field can be modified only when the RX_SECTION_EN bit and the RX_FC_SECTION_EN bit in the MSF_CLOCK CONTROL register are disabled	RW	0x7

5.7.14 MSF_DLL_DATA_DELAY_CTL

The msf_dll_data_delay_ctl register (rev B only) is useful when running in static alignment mode. In this mode, the receiver samples incoming data at the 90 degree point relative to the clock based on the assumption that the data and clock are very precisely matched as they enter IXP2800. Normally, the default setting is adequate. However, systems designers may notice that dynamic training cycles yield phasemon register values that suggest the 90 degree point is not the nominal deskew setting for the majority of the incoming data bits. When that is the case, the internal skew between data and clocks can be adjusted by programming this register to advance or delay the data relative to the clock, thus bringing it closer to the 90 degree sampling point.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																												MSF_DATA_DELAY_SELECT			

Bits	Field	Description	RW	Reset
[31:2]	RESERVED	Reserved	W	0x0
[1:0]	MSF_DATA_DELAY_SELECT	00 Nominal delay is inserted on the msf receive data bits. 01 Nominal delay + 50psec is inserted on the msf receive data bits. 10 Nominal delay - 50psec is inserted on the msf receive data bits. 11 Nominal delay - 100psec is inserted on the msf receive data bits. This field can be modified only when the RX_SECTION_EN bit and the RX_FC_SECTION_EN bit in the MSF_CLOCK CONTROL register are disabled	W	0x0

5.7.15 FC_DYNFILT_THRESH

The FC_DYNFILT_THRESH register defines the number of mismatched samples that must be seen before an adjustment is made to advance or delay the sampling clock phase. See [Section 5.7.13](#) for an explanation of dynamic jitter compensation.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																												#_SAMPLES			

Bits	Field	Description	RW	Reset
[31:3]	RESERVED	Reserved	RO	0
[2:0]	_SAMPLES	The number of mismatched samples that must be seen before an adjustment is made to advance or delay the sampling clock phase. Legal values are 1-4. This field can be modified only when the RX_FC_SECTION_EN bit in the MSF_CLOCK CONTROL register is disabled	RW	0x7

5.7.16 FC_DLL_DATA_DELAY_CTL

The `fc_dll_data_delay_ctl` register (rev B only) is useful when running in static alignment mode. In this mode, the receiver samples incoming data at the 90 degree point relative to the clock based on the assumption that the data and clock are very precisely matched as they enter IXP2800.

Normally, the default setting is adequate. However, systems designers may notice that dynamic training cycles yield phasemon register values that suggest the 90 degree point is not the nominal deskew setting for the majority of the incoming data bits. When that is the case, the internal skew between data and clocks can be adjusted by programming this register to advance or delay the data relative to the clock, thus bringing it closer to the 90 degree sampling point.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																												FC_DATA_DELAY_SELECT		

Bits	Field	Description	RW	Reset
[31:2]	RESERVED	Reserved	W	0x0
[1:0]	FC_DATA_DELAY_SELECT	00 Nominal delay is inserted on the msf receive data bits. 01 Nominal delay + 50psec is inserted on the msf receive data bits. 10 Nominal delay - 50psec is inserted on the msf receive data bits. 11 Nominal delay - 100psec is inserted on the msf receive data bits. This field can be modified only when the RX_SECTION_EN bit and the RX_FC_SECTION_EN bit in the MSF_CLOCK CONTROL register are disabled	W	0x0

5.7.17 HWM_CONTROL

This register is used to control high water marks for RBUF, FCEFIFO, and FCIFIFO. This register must not be changed while SPI-4 or CSIX CFrames are being received.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																		FCIFIFO_EXT_HWM	FCIFIFO_INT_HWM	FCEFIFO_HWM	RBUF_S_HWM	RBUF_D_HWM	RBUF_C_HWM							

Bits	Field	Description	RW	Reset
[31:12]	RESERVED	Reserved	RO	0
[11:10]	FCIFIFO_Ext_HWM	Flow Control Ingress FIFO High Watermark for external use. Simplex mode: Deassert FC_Ingress_Status[TM_CReady] if the number of CWord entries used in the FCIFIFO is >= the programmed high watermark. Note this is different than the IXP2400. Full duplex mode: Assert RXCFC to the Egress IXP2800 if the number of CWord entries used in the FCIFIFO is >= the programmed high watermark. 0x00: 32 CWords 0x01: 64 CWords 0x10: 128 CWords 0x11: 192 CWords	RW	0
[9:8]	FCIFIFO_Int_HWM	Flow Control Ingress FIFO High Watermark for internal use. Assert FCI_Near_Full to the MEs if the number of CWord entries used in the FCIFIFO is >= the programmed water mark. Note this is different than the IXP2400. 0x00: 16 CWords 0x01: 32 CWords 0x10: 64 CWords 0x11: 128 CWords	RW	0
[7:6]	FCEFIFO_HWM	Flow Control Egress FIFO High Watermark. Near full if the number of occupied CWords is >= number in this field. Controls when FC_Egress_Status[FCEFIFO_Full] asserts. Note this is different than the IXP2400. 0x00:16 CWords 0x01: 32 CWords 0x10: 64 CWords 0x11: 128 CWords	RW	0

Bits	Field	Description	RW	Reset
[5:4]	RBUF_S_HWM	RBUF SPI-4 High Watermark. Near full if the number of occupied RBUF entries allocated for SPI-4 data is \geq number in this field. The number of entries is a function of the number of partitions and entry size as programmed in MSF_Rx_Control. Refer to Table 5-45 and Table 5-48 . Note this is different than the IXP2400.	RW	0
[3:2]	RBUF_D_HWM	RBUF Data High Watermark. Near full if the number of occupied RBUF entries allocated for CSIX Data CFrames is \geq number in this field. The number of entries is a function of the number of partitions and entry size as programmed in MSF_Rx_Control. Refer to Table 5-45 and Table 5-48 . Note this is different than the IXP2400.	RW	0
[1:0]	RBUF_C_HWM	RBUF Control High Watermark. Near full if the number of occupied RBUF entries allocated for CSIX Control CFrames is \geq number in this field. The number of entries is a function of the number of partitions and entry size as programmed in MSF_Rx_Control. Refer to Table 5-45 and Table 5-48 . Note this is different than the IXP2400.	RW	0

[Table 5-48](#) shows the HWM by number of RBUF elements. Refer to [Table 5-31](#) to determine how many RBUF elements exist for each partition.

Table 5-48. RBUF High Water Marks

Number of Elements in Partition	Value Programmed in HWM Field (Note -- HWM indicates that \geq number of entries indicated are occupied)			
	00 (1/4 Entries)	01 (1/2 Entries)	10 (3/4 Entries)	11 (7/8 Entries)
128	32	64	96	112
96	24	48	72	84
64	16	32	48	56
48	12	24	36	42
32	8	16	24	28
24	6	12	18	21
16	4	8	12	14
12	3	6	9	10
8	2	4	6	7
4	1	2	3	3

5.7.18 RX_THREAD_FREELIST_# (# = 0,1,2)

Microengines write to these registers to add a Context to RX_THREAD_FREELIST_#. The various Freelists are connected to protocol type as shown in [Table 5-49](#). The size of the Freelists are:

- freelist 0 - 64 entries
- freelist 1 - 32 entries
- freelist 2 - 32 entries

Users should not place more entries on the list than the list can hold. Note that the Microengine number is specified differently for the IXP2400.

Refer to the IXP2800 Hardware Reference Manual (HRM) for definition of the Element status.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED															SIG_NO			ME_CLUS	RESERVED	ME_NO			THD		XFER_REG						

Bits	Field	Description	RW	Reset
[31:16]	RESERVED	Reserved	RO	0
[15:12]	SIG_NO	Which signal to deliver when pushing the Element status.	WO	0
[11]	ME_CLUS	ME Cluster	WO	0
[10]	RESERVED	Reserved	RO	0
[9:7]	ME_NO	Microengine number that will be signaled. Valid IXP2800 ME numbers are 0 - 7. Valid IXP2400 ME numbers are 0 - 3.	WO	0
[6:4]	THD	Indicates which thread will get the RBUF Element status pushed to it.	WO	0
[3:0]	XFER_REG	Indicates which S_Transfer registers will get the RBUF Element status pushed to it. Two consecutive registers, starting with the one in this field, are written.	WO	0

Table 5-49. Rx_Thread_Freelist Use

Number of Partitions ¹	Use	CSIX_Freelist ²	Rx_Thread_Freelist_# Used		
			0	1	2
1	SPI-4 only	n/a	SPI-4 Ports equal to or below Rx_Port_Map	SPI-4 Ports above Rx_Port_Map	Not Used
2	CSIX only	0	CSIX Data	CSIX Control	Not Used
		1	CSIX Data and CSIX Control	Not Used	Not Used
3	Both SPI-4 and CSIX	0	CSIX Data	SPI-4	CSIX Control
		1	CSIX Data and CSIX Control	SPI-4	Not Used

1. Programmed in MSF_Rx_Control[RBUF_Partition].
2. Programmed in MSF_Rx_Control[CSIX_Freelist].

5.7.19 RX_PORT_MAP

This register is used to connect SPI-4 ports to Contexts by controlling which RX_THREAD_FREELIST_# each port uses. This is only used when MSF_RX_CONTROL[RBUF_PARTITION] is set for one SPI-4 partition. This register must not be changed while SPI-4 data is being received.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						PORT_NO								

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	PORT_NO	Mpackets from SPI-4 ports equal to or less than this number will be pushed to Contexts from RX_THREAD_FREELIST_0. Mpackets from SPI-4 ports above this number will be pushed to Contexts from RX_THREAD_FREELIST_1. To have all mpackets pushed to Contexts from RX_THREAD_FREELIST_0 put 0xFF into this field.	RW	FF

5.7.20 RBUF_ELEMENT_DONE

This register is written with an RBUF element number and the number is placed onto a freelist so that the RBUF element can be reused. It is illegal to write this register with the element number of an already free element.

After Reset, all RBUF elements are free.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						ELE_TO_FREE								

Bits	Field	Description	RW	Reset
[31:7]	RESERVED	Reserved	RO	0
[6:0]	ELE_TO_FREE	Indicates the number of the element to free.	WO	undef

5.7.21 RX_CALENDAR_LENGTH

This register is used to set the length of the SPI-4 RSTAT calendar. This register must not be changed while SPI-4 data is being received.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																						LENGTH									

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	LENGTH	Length of the RX_CALENDAR. The value represents the number of time slots allotted for the 2-bit status messages sent out in each RSTAT message. For example, a value of 0x1 indicates one 2-bit status message per RSTAT message. The one exception is that a value of 0x0 indicates 256 2-bit status messages. Each 2-bit message indicates the status of the RBUF at the time the 2-bits are transmitted on the RSTAT pins. In other words, the value of the 2-bit messages may change in the middle of an RSTAT message.	RW	0

5.7.22 FCEFIFO_VALIDATE

This register is used to validate a CFrame written into FCEFIFO by software. The CFrame will not be transmitted on TXCDAT until it is validated. The data is not used; any write to this register does the validate.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
ANY_DATA																															

Bits	Field	Description	RW	Reset
[31:0]	ANY_DATA	The act of writing to this register validates the CFrame written into FCEFIFO. Any data may be written to this register.	WO	undef

5.7.23 TX_SEQUENCE_# (# = 0,1,2)

A read of this register provides a wrapping count of the number of TBUF elements that have been transmitted from the TBUF the partition. The number of partitions in use is based on MSF_TX_CONTROL[TBUF_PARTITION]. The count advances when the entire content of the element have been transmitted, or if the element control word skip bit was set (i.e., so it is safe to write new data into the element).

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
EMPTY	RESERVED																							SEQUENCE							

Bits	Field	Description	RW	Reset
[31]	EMPTY	Indicates if there are any valid elements (i.e. elements that are ready to be transmitted) for this partition. 0: One or more elements are valid. 1: No elements are valid.	RO	1
[30:8]	RESERVED	Reserved	RO	0
[7:0]	SEQUENCE	Sequence count of how many elements have been transmitted for this partition. The count always counts from 0 to 255 and then wraps back to 0, regardless of the number of elements in the partition.	RO	0

5.7.24 RX_THREAD_FREELIST_TIMEOUT_# (# = 0,1,2)

There is one RX_THREAD_FREELIST_TIMEOUT register associated with each RX_THREAD_FREELIST.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED															TIMEOUT_INTERVAL																

Bits	Field	Description	RW	Reset
[31:13]	RESERVED	Reserved	RO	0
[12:0]	TIMEOUT_INTERVAL	This is the number of internal bus clocks (1/2 Microengine frequency), starting from the autopush of a null or non-null Receive Status Word, that are allowed to elapse before triggering the autopush of a null Receive Status Word. Every time any Receive Status Word is autopushed the timer is reset and restarted. If no new receive traffic has come in and the time-out interval has been reached, hardware will automatically autopush a null Receive Status Word to the next thread in the RX_THREAD_FREELIST. A value of 0 means that the timer is disabled.	RW	0

5.7.25 RX_PORT_CALENDAR_STATUS_# (0 TO 255)

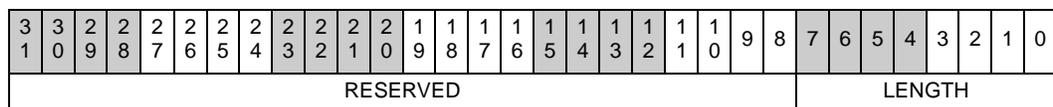
Rev B -- These registers allow software to control the SPI-4 Calendar Status driven out on RSTAT[1:0] or TXCDAT[1:0]. Each register controls the calendar status value for a different port, corresponding to the port number (indicated by #).

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																											STATUS				

Bits	Field	Description	RW	Reset
[31:2]	RESERVED	Reserved	RO	0
[1:0]	PORT_CALENDAR STATUS	<p>The value driven as the status for a given port is dependent on the MSF_RX_CONTROL[Rx_Calendar_Mode] (Section 5.7.1) setting.</p> <p>When in Conservative_Value Mode, the value for each port is the most conservative of:</p> <ul style="list-style-type: none"> the RBUF status based on RBUF high-water-mark (Section 5.7.17) RSTAT_Ov_Value (if RSTAT_Override is 1) (Section 5.7.1) the value in Rx_Port_Calendar_Status_# for the port. <p>Satisfied is more conservative than Hungry, which is more conservative than Starving. This allows software to give a more conservative value for all ports at once (RSTAT_Override), or control each port individually (Rx_Port_Calendar_Status_#), or allow all ports to be controlled by the RBUF hardware high-water-mark control. In this mode it is illegal to program 11 into this field. To have no override, set this to Starving, which is the least conservative and therefore won't have any effect.</p> <p>When in Force_Override Mode, the value for each port is the value in this field with one exception. - The value of 11 is the calendar framing indicator, so can't be used as a calendar value. Programming the value of 11 in the register is an indication to not override, but instead use the more conservative of RBUF status and RSTAT_Ov_Value (if RSTAT_Override is 1) as the calendar value for this port.</p>	RW	0

5.7.26 TX_CALENDAR_LENGTH

Used to set the length of the SPI-4 TSTAT calendar. This register must not be changed while SPI-4 data is being received.



Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	LENGTH	Length of TX_CALENDAR. The value represents the number of entries to use in TX_CALENDAR. For example an value of 0x1 indicates 1 entry in the calendar. The one exception is that a value of 0x0 indicates 256 entries.	RW	0

5.7.27 TX_CALENDAR_# (# = 0 - 255)

TX_CALENDAR is a RAM with 256 entries, each of which is a time slot in the calendar. The address number corresponds to the time slot. Each entry has the following format.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
RESERVED																						PORT#									

Bits	Field	Description	RW	Reset
[31:8]	RESERVED	Reserved	RO	0
[7:0]	PORT#	Port number for the time slot. In normal operation this register is write only. It can be read, for test purposes, but only if MSF_TX_CONTROL[TSTAT_EN] is a 0.	WO	Undef

5.7.28 TX_PORT_STATUS_# (# = 0 - 255)

TX_PORT_STATUS_# represents 256 registers, one for each of 256 SPI-4 ports. The port status is updated each time a new calendar status for that port is received, according to the mode programmed in TX_CONTROL[TX_STATUS_UPDATE_MODE]. When the register is read, the Status field is replaced by the value selected by TX_CONTROL[TX_STATUS_READ_MODE]. Each entry has the following format.

Note that this set of registers or the TX_MULTIPLE_PORT_STATUS set of registers must be read by the software in order to determine the status of each port and send data to them accordingly. The MSF hardware does not check these registers for port status before sending data out to a particular port.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
RESERVED																											STATUS				

Bits	Field	Description	RW	Reset
[31:2]	RESERVED	Reserved	RO	Undef
[1:0]	STATUS	Status value for port.	RO	0x3

5.7.29 TX_MULTIPLE_PORT_STATUS_# (# = 0 - 15)

TX_MULTIPLE_PORT_STATUS_# provides the same information as TX_PORT_STATUS but in a denser format. There are 16 addresses, each reading 16 ports. Each entry has the following format. The value n in the description is replaced by 0, 16, 32, etc for each of the 16 registers, starting with the lowest address register (TX_MULTIPLE_PORT_STATUS_0).

Bits	Field	Description	RW	Reset
[31:17]	RESERVED	Reserved	RO	0
[16]	FORCE_CDEAD	Force sending Dead Cycle on TDAT pins. 0—No effect. 1—Send back to back Dead Cycles. On assertion of this bit the current CFrame transmission is completed before beginning the Dead Cycles.	RW	0
[15:10]	ALPHA	Number of repetitions of training sequence. The value written indicates how many repeats of the 20-word pattern to do (it represents the α value in the SPI-4 spec). A value of 0x0 indicates 1 and a value of 0x3F indicates 64.	RW	0
[9]	SING_TRAIN	Single Train. Transmit training pattern once on signals timed to TCLK. 0—No effect. 1—Transmit training pattern. On assertion of this bit the current element transmission is completed before beginning transmission of the training pattern.	WO	0
[8]	CONT_TRAIN	Continuously Train. Transmit training pattern continuously on signals timed to TCLK. 0—No effect. 1—Transmit training pattern. On assertion of this bit the current element transmission is completed before beginning the training pattern. On deassertion of this bit the current training pattern sequence (as specified by Alpha field) is completed before resuming normal transmit operation.	RW	0
[7]	DEAD_EN_FCIDLE	Enable to automatically transmit Dead Cycles on signals timed to TCLK. 0—Ignore TRAIN_FLOW_CONTROL[DETECT_FCIDLE] bit. 1—Transmit Dead Cycles while TRAIN_FLOW_CONTROL[DETECT_FCIDLE] bit is asserted. On assertion of that bit the current element transmission is completed before beginning transmission of the Dead Cycles.	RW	0
[6]	TRAIN_EN_FCDEAD	Enable automatically transmit training pattern on signals timed to TCLK. 0—Disable transmit training on the TDAT pins (Ignore the TRAIN_FLOW_CONTROL[DETECT_FCDEAD] bit). 1—Perform transmit training pattern while TRAIN_FLOW_CONTROL[DETECT_FCDEAD] bit is asserted. On assertion of the DETECT_FCDEAD bit, the current element transmission is completed before beginning transmission of the training pattern. On deassertion of either this bit or DETECT_FCDEAD bit, the current training pattern is completed before resuming normal transmit operation. Training does not necessarily begin on assertion of the respective TRAIN_FLOW_CONTROL[DETECT_FCDEAD] bit. If the training pattern does not begin because it is waiting for the current transmission to complete, and during that time the TRAIN_FLOW_CONTROL[DETECT_FCDEAD] bit is desasserted, then training may never be sent.	RW	0

Bits	Field	Description	RW	Reset
[5]	DETECT_CDEAD	<p>RDAT pins status.</p> <p>This bit dynamically changes as dead cycles are received on the RDAT pins.</p> <p>0—RDAT input has received Dead Cycles for less than consecutive cycles. 1—RDAT input has received Dead Cycles for 2 or more consecutive cycles.</p>	RO	0
[4]	DETECT_CIDLE	<p>RDAT pins status.</p> <p>0—RDAT input has received less than two consecutive Idle CFrames. 1—RDAT input has received 2 or more consecutive Idle CFrames.</p>	RO	0
[3]	DETECT_NO_CAL	<p>Status channel disabled indicator. Status channel is either TSTAT or RXCDAT depending on MSF_Tx_Control[TSTAT_Select]. When LVTTTL is used this bit indicates TSTAT input has had framing pattern for more than 32 consecutive cycles; when LVDS is used this bit indicates that 3 consecutive training patterns were detected on RXCDAT.</p> <ul style="list-style-type: none"> 0—Status channel indicates calendar is active. 1—Status channel indicates need for training (i.e. no calendar active). 	RO	0
[2]	TRAIN_EN_TSTAT	<p>Enable to automatically transmit training pattern on signals timed to TCLK.</p> <p>0—Ignore Detect_No_Calendar bit. 1—Transmit training pattern while Detect_No_Calendar bit is asserted.</p> <p>On assertion of DETECT_NO_CAL bit the current element transmission is completed before beginning transmission of the training pattern. On deassertion of either this bit or Detect_No_Calendar bit the current training pattern is completed before resuming normal transmit operation.</p>	RW	0
[1]	RSTAT_EN	<p>RSTAT Enable.</p> <p>0—RSTAT is held statically at 0x3. 1—RSTAT is used to send FIFO status according to the description in SPI-4 Receive Flow Control.</p> <p><i>Note:</i> enables writing TXCDAT in LVDS mode.</p>	RW	0
[0]	IGN_TRAIN	<p>Prevents automatic deskew training on the RDAT, RCTL, RPAR, RPROT pins.</p> <p>0—Automatically perform training when the training pattern is received. 1—Ignore training patterns when received.</p> <p>This field can be modified only when the RX_FC_SECTION_EN and RX_SECTION_EN bits in the MSF_CLOCK CONTROL register are disabled</p>	RW	0

5.7.32 TRAIN_CALENDAR

This register is used for control and status related to pin deskew training. Train_Calendar is for pins controlled by RXCCLK/TXCCLK when used for LVDS SPI-4 status channel.

Training is sent at scheduled times under software control. Software can detect an excessively long training pattern on RDAT and send a training sequence in response.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED											ALPHA						SING_TRAIN	CONT_TRAIN	RESERVED						IGN_TRAIN		

Bits	Field	Description	RW	Reset
[31:16]	RESERVED	Reserved	RO	0
[15:10]	ALPHA	Number of repetitions of training sequence. The value written indicates how many repeats of the 20-word pattern to do (it represents the α value in the SPI-4 spec). A value of 0x0 indicates 1 and a value of 0x3F indicates 64.	RW	0
[9]	SING_TRAIN	Transmit training pattern Train_Calendar[Alpha] times on signals timed to TXCCLK. <ul style="list-style-type: none"> 0—No effect. 1—Transmit training pattern. On assertion of this bit the current calendar transmission is completed before beginning the training pattern. 	WO	0
[8]	CONT_TRAIN	Transmit training pattern continuously on signals timed to TXCCLK. <ul style="list-style-type: none"> 0—No effect. 1—Transmit training pattern. On assertion of this bit the current calendar transmission is completed before beginning transmission of the training pattern. On deassertion of this bit the current training pattern sequence (as specified by Alpha field) is completed before resuming normal transmit operation. 	RW	0
[7:1]	RESERVED	Reserved	RO	
[0]	IGN_TRAIN	Prevents automatic deskew training on the RXCDAT pin. <ul style="list-style-type: none"> 0—Automatically perform training when the training pattern is received. 1—Ignore training patterns when received. 	RW	0

5.7.33 TRAIN_FLOW_CONTROL

This register is used for control and status related to pin deskew training. Train_Flow_Control is for pins controlled by RXCCLK/TXCCLK when used for CSIX Flow Control. It is possible for conditions to exist that simultaneously force fc training, dead cycles, and/or idle cycles. In that event the following rules apply:

- For Simplex configurations, the precedence is FC Training cycles (Highest), Dead cycles, then Idle cycles.
- For Duplex configurations, the precedence is Idle cycles (Highest), Dead cycles, then FC Training cycles."

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0		
RESERVED													FORCE_FCDEAD	FORCE_FCIDLE	ALPHA						SING_TRAIN	CONT_TRAIN	RESERVED	TD_EN_CDEAD	DET_FCDEAD	DET_FCIDLE	DET_TXCFC_SUS	TRAIN_EN_CFC	RXCFC_EN	IGN_TRAIN

Bits	Field	Description	RW	Reset
[31:18]	RESERVED	Reserved	RO	0
[17]	FORCE_FCDEAD	Force sending Dead Cycle on TXC pins. 0—No effect. 1—Send back to back Dead Cycles. On assertion of this bit the current CFrame transmission is completed before beginning the Dead Cycles. This bit and Force_Idle must not both be set.	RW	0
[16]	FORCE_FCIDLE	Force sending Idle CFrames on TXC pins. 0—No effect. 1—Send back to back Idle CFrames. On assertion of this bit the current CFrame transmission is completed before beginning the Idle CFrames. This bit and Force_Dead must not both be set.	RW	0
[15:10]	ALPHA	Number of repetitions of training sequence. The value written indicates how many repeats of the 20-word pattern to do (it represents the α value in the SPI-4 spec). A value of 0x0 indicates 1 and a value of 0x3F indicates 64.	RW	0
[9]	SING_TRAIN	Transmit training pattern once on signals timed to TXCCLK. 0—No effect. 1—Transmit training pattern. On assertion of this bit the current CFrame transmission is completed before beginning the training pattern.	WO	0

Bits	Field	Description	RW	Reset
[8]	CONT_TRAIN	Transmit training pattern continuously on signals timed to TXCCLK. 0—No effect. 1—Transmit training pattern. On assertion of this bit the current element transmission is completed before beginning the training pattern. On deassertion of this bit the current training pattern sequence (as specified by Alpha field) is completed before resuming normal transmit operation.	RW	0
[7]	RESERVED	Reserved	RO	0
[6]	TD_EN_CDEAD	Enable to automatically force TXC pins. 0—Disable Training on the TXC pins (Ignore the TRAIN_DATA[DETECT_CDEAD] bit). 1— Perform training on the TXC pins according to Duplex Mode. If Full Duplex Mode Transmit Dead Cycles while TRAIN_DATA[DETECT_CDEAD] bit is asserted. On assertion of that bit the current element transmission is completed before beginning transmission of the dead cycles. If Simplex Mode Transmit training pattern while TRAIN_DATA[DETECT_CDEAD] bit is asserted. On assertion of the DETECT_CDEAD bit, the current element transmission is completed before beginning transmission of the training pattern. On deassertion of either this bit or DETECT_CDEAD bit, the current training pattern is completed before resuming normal transmit operation. Training does not necessarily begin on assertion of the TRAIN_DATA[DETECT_CDEAD] bit. If the training pattern does not begin because it is waiting for the current transmission to complete, and during that time the TRAIN_DATA[DETECT_CDEAD] bit is desasserted, then training may never be sent.	RW	0
[5]	DET_FCDEAD	RXCDAT pins status. This bit dynamically changes as dead cycles are received on the RXCDAT pins. 0—RXCDAT input has received Dead Cycles for less than 2 consecutive cycles. 1—RXCDAT input has received Dead Cycles for 2 or more consecutive cycles.	RO	0
[4]	DET_FCIDLE	RXCDAT pins status. 0—RXCDAT input has received less than two consecutive Idle CFrames. 1—RXCDAT input has received 2 or more consecutive Idle CFrames.	RO	0
[3]	DET_TXCFC_SUS	TXCFC link disabled indicator. 0—TXCFC input has been asserted for 32 or less consecutive cycles. 1—TXCFC input has been asserted for more than 32 consecutive cycles.	RO	0

Bits	Field	Description	RW	Reset
[2]	TRAIN_EN_CFC	Enable to automatically transmit training pattern. 0—Ignore DETECT_TXCFC_SUSTAINED bit. 1—Transmit training pattern while DETECT_TXCFC_SUSTAINED bit is asserted. On assertion of DETECT_TXCFC_SUSTAINED bit the current CFrame transmission is completed before beginning the training pattern. On deassertion of either this bit or DETECT_TXCFC_SUSTAINED bit the current training pattern is completed before resuming normal transmit operation.	RW	0
[1]	RXCFC_EN	RXCFC Enable. 0—RXCFC is held statically asserted. 1—RXCFC is used to send FCIFIFO status.	RW	0
[0]	IGN_TRAIN	Prevents automatic deskew training on the RXCDAT, RXCSOF, RXCPAR pins. 0—Automatically perform training when the training pattern is received. 1—Ignore training patterns when received	RW	0

5.7.34 RX_PHASEMON_# (# = pin name)

The RX_PHASEMON_# registers allow the user to monitor which dll output clock phase is being used to sample the data. The sampling phase can be set by software, by deskew training cycles, or by the dynamic deskew adjust circuitry.

Table 5-54. List of RX_PHASEMON_# Registers

RX_PHASEMON_# - Replace # With the Following			
RDAT0	RDAT7	RDAT14	RXCDAT1
RDAT1	RDAT8	RDAT15	RXCDAT2
RDAT2	RDAT9	RCTL	RXCDAT3
RDAT3	RDAT10	RPAR	RXCPAR
RDAT4	RDAT11	RPROT	RXCSRB
RDAT5	RDAT12	RXCOSOF	
RDAT6	RDAT13	RXCDAT0	

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0									
RESERVED																								TR_STATUS	CLK_PHASE					



Bits	Field	Description	RW	Reset
[31:5]	RESERVED	Reserved	RO	0
[4]	TR_STATUS	Training Status: a value of 1 means a failure occurred on this bit during the last training sequence received. Most likely cause of this failure is that the training edge was skewed too far from the other signals in the training group, or no edge occurred at all.	RO	0
[3:0]	CLK_PHASE	This field dynamically indicates which of the 16 dll output clocks is being used to sample the most significant bit of the incoming data. The clock phase 180 degrees from this phase is used to sample the least significant bit of the incoming data. Note that each increment is 22.5 degrees. Value - Degree dll clock phase 0x0 - 90 degree 0x1 - 112.5 degree 0x2 - 135 degree 0x3 - 157.5 degree 0x4 - 180 degree 0x5 - 202.5 degree 0x6 - 225.0 degree 0x7 - 247.5 degree 0x8 - 270 degree 0x9 - 292.5 degree 0xA - 315 degree 0xB - 337.5 degree 0xC - 0 degree 0xD - 22.5degree 0xE - 45 degree 0xF - 67.5 degree	RO	0

5.7.35 MSF_IO_BUF_CTL

This register is used for RCOMP control for the MSF Interface.

3	3	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0										
RESERVED											MSF_RCOMP_FB_READ					MSF_RCOMP_VCOMPL_OUT	MSF_RCOMP_OUT	MSF_RCOMP_READ					MSF_SW_WAROUND		MSF_DYN_DESKEW_DIS		MSF_RCOMP_VALUE					MSF_RCOMP_OVERRIDE	MSF_PRE_EQ_VALUE				

Bits	Field	Description	RW	Reset
[31:25]	RESERVED	Reserved	RO	undef
[24:20]	MSF_RCOMP_FB_READ	This is the 5-bit binary value (converted from a 25-bit thermometer code) generated by the MSF receivers' auto-Rcomp circuit (shared by all MSF - actual control back to Rcomp (in feedback)	RO	undef
[19]	MSF_RCOMP_VCOMPL_OUT	This signal indicates the status of the 1.0V comparator output of the auto-Rcomp circuit (shared by all MSF LVDS receiver pins)	RO	undef
[18]	MSF_RCOMP_OUT	This signal indicates the status of the 1.4V comparator output of the auto-Rcomp circuit (shared by all MSF LVDS receiver pins)	RO	undef
[17:13]	MSF_RCOMP_READ	This is the 5-bit binary value (converted from a 25-bit thermometer code) generated by the MSF receivers' auto-Rcomp circuit (shared by all MSF LVDS receiver pins)	RO	undef
[12]	MSF_SW_WAROUND	Select the external control for the internal 100-ohm resistance (workaround mode - interfere with the feedback)	RW	undef
[11]	MSF_DYN_DESKEW_DIS	0 - MSF dynamic de-skew (in receiver) enabled (reset) 1 - MSF dynamic de-skew (in receiver) disabled This field can be modified only when the RX_SECTION_EN and RX_FC_SECTION_EN bits in the MSF_CLOCK CONTROL register are disabled	RW	undef

Bits	Field	Description	RW	Reset
[10:6]	MSF_RCOMP_VALUE	This 5-bit binary value (converted to a 25-bit thermometer code) determines the MSF receivers' Rcomp setting - needs bit[5] to be set (shared by all MSF LVDS receiver pins)	RW	undef
[5]	MSF_RCOMP_OVERRIDE	0 - MSF auto-Rcomp is applied to receiver (reset) 1 - Rcomp value programmed in bits [10:6] is applied to MSF LVDS receivers (i.e. auto-Rcomp setting is ignored)	RW	undef
[4:0]	MSF_PRE_EQ_VALUE	Sets the pre-equalization amount to be set for the MSF LVDS drivers. 0 - Pre-equalization is off Any non-zero value - Corresponding pre-equalization is applied to output signals.	RW	undef

5.7.36 FC_IO_BUF_CTL

This register is used for RCOMP control for the MSF Interface.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED												FC_RCOMP_FB_READ		FC_RCOMP_VCOMPL_OUT	FC_RCOMP_OUT	FC_RCOMP_READ		FC_SW_WAROUND		FC_DYN_DESKEW_DIS	FC_RCOMP_VALUE		FC_RCOMP_OVERRIDE	FC_PRE_EQ_VALUE					

Bits	Field	Description	RW	Reset
[31:25]	RESERVED	Reserved	RO	undef
[24:20]	FC_RCOMP_FB_READ	This is the 5-bit binary value (converted from a 25-bit thermometer code) generated by the FC receivers' auto-Rcomp circuit (shared by all FC - actual control back to Rcomp (in feedback)	RO	undef
[19]	FC_RCOMP_VCOMPL_OUT	This signal indicates the status of the 1.0V comparator output of the auto-Rcomp circuit (shared by all FC LVDS receiver pins)	R0	undef
[18]	FC_RCOMP_OUT	This signal indicates the status of the 1.4V comparator output of the auto-Rcomp circuit (shared by all MSF LVDS receiver pins)	RO	undef
[17:13]	FC_RCOMP_READ	This is the 5-bit binary value (converted from a 25-bit thermometer code) generated by the FC receivers' auto-Rcomp circuit (shared by all FC LVDS receiver pins)	RO	undef

Bits	Field	Description	RW	Reset
[12]	FC_SW_WAROUND	Select the external control for the internal 100-ohm resistance (workaround mode - interfere with the feedback)	RW	undef
[11]	FC_DYN_DESKEW_DIS	0 - FC dynamic de-skew (in receiver) enabled (reset) 1 - FC dynamic de-skew (in receiver) disabled This field can be modified only when the RX_SECTION_EN bit and RX_FC_SECTION_EN bit in the MSF_CLOCK CONTROL register are disabled	RW	undef
[10:6]	FC_RCOMP_VALUE	This 5-bit binary value (converted to a 25-bit thermometer code) determines the FC receivers' Rcomp setting - needs bit[5] to be set (shared by all FC LVDS receiver pins)	RW	undef
[5]	FC_RCOMP_OVERRIDE	0 - FC auto-Rcomp is applied to receiver (reset) 1 - Rcomp value programmed in bits [10:6] is applied to FC LVDS receivers (i.e. auto-Rcomp setting is ignored)	RW	undef
[4:0]	FC_PRE_EQ_VALUE	Sets the pre-equalization amount to be set for the FC LVDS drivers. 0 - Pre-equalization is off Any non-zero value - Corresponding pre-equalization is applied to output signals.	RW	undef

5.7.37 MSF Initial Setup Procedure for the IX2800 Rev A

This section explains the CSR settings required to make the MSF ready for operation.

1. Program the Clock Control Register with the desired clock ratio. If TCLK is being generated internally, the ratio should be microengine frequency / MSF bus frequency. If TCLK is taken from TCLK_REF, the MSF frequency will be the same as the input clock.
2. Clear bit 7 of IXP_RESET0 to bring the MSF out of reset.
3. Reset timer columns in test debug registers. Write a 1 and then a 0 to bit 10 of MSF CSRs 0x80f4-0x8104. Only required for A step devices.
4. Initialize SPI-4 calendar if needed. TX_CALENDAR_LENGTH, TX_CALENDAR_#, RX_CALENDAR_LENGTH
5. Initialize the lower 28 bits of MSF_RX_CONTROL and MSF_TX_CONTROL, this value will vary per implementation.
6. Write 0x10 into MSF_CLOCK_CONTROL, this enables the output of TCLK.
7. Initialize devices connected to MSF and enable their clocks that serve as inputs to the IXP2800 and wait for RCLK to stabilize.
8. Write 0x1490 into MSF_CLOCK_CONTROL, this selects the RCLK input, enables the RCLK DLL and enables the RCLK_REF output clock.
9. Wait 1mS for RCLK DLL to lock.
10. If you are not using the flow control bus, skip forward to step 15
11. Write 0x14D0 into MSF_CLOCK_CONTROL, this enables the TXCCLK output. The output clock is selected by MSF_TX_CONTROL[TXCCLK_SOURCE].

12. Start RXCCLK and wait for it to stabilize.
13. Write 0x3CD0 into MSF_CLOCK_CONTROL, this selects the RXCCLK input, enables the RXCCLK DLL.
14. Wait 1mS for RXCCLK DLL to lock.
15. OR 0x32F into MSF_CLOCK_CONTROL (or an applicable subset), in this case this enables the receive and transmit calendars, enables the RSCLK output and enables the MSF and Flow Control receive and transmit sections.
16. Train all interfaces as described in HRM section 8.6.3.
17. Initialize CSIX_TYEMAP, HWM_CONTROL, RX_PORTMAP, and RX_THREAD_FREELIST_TIMEOUT with values applicable to your application.
18. If using CSIX, enable flow control as described in HRM section 8.7.
19. OR in the appropriate enables in the upper 4 bits of MSF_RX_CONTROL and MSF_TX_CONTROL, this enables the SPI4/CSIX receive and transmit.
20. Clear all MSF Interrupts by writing 0xFFFFFFFF to MSF_INTERRUPT_STATUS.
21. Enable appropriate MSF interrupts by initializing MSF_INTERRUPT_ENABLE
22. Enable microengine receive threads and begin frame processing.

5.7.38 MSF Initial Setup Procedure for the IX2800 Rev B

Perform the following steps to initialize the MSF interface for a B stepping:

1. Program the Clock Control Register with the desired clock ratio. If TCLK is generated internally, the ratio should be microengine frequency / MSF bus frequency. If TCLK is taken from TCLK_REF, the MSF frequency will be the same as the input clock.
2. Clear bit 7 of IXP_RESET0 to bring the MSF out of reset.
3. If needed, initialize the SPI-4 calendar, using TX_CALENDAR_LENGTH, TX_CALENDAR_#, RX_CALENDAR_LENGTH.
4. Initialize the lower 28 bits of MSF_RX_CONTROL and MSF_TX_CONTROL; this initialization value depends on the implementation. *Note:* In the B stepping, the RSTAT_OV_VALUE bits [19:18], RSTAT_OVERRIDE bit [17], and RX_CALENDAR_MODE bit [11], registers have been modified and must be programmed appropriately for the desired calendar mode. Additionally, RX_PORT_CALENDAR_STATUS_# must be programmed with the desired status value for each port. Refer to the *Intel® IXP2800 Network Processor Programmers Reference Manual* for more information.
5. Write 0x10 into MSF_CLOCK_CONTROL to enable the output of TCLK; [Figure 3](#) begins with this step.
6. Initialize devices connected to the MSF and enable their clocks that serve as inputs to the Intel® IXP2800; wait for RCLK to stabilize.
7. Write 0x1010 into MSF_CLOCK_CONTROL to take the MSF DLL out of reset. *Note:* the RCLK input clock must be present and stable before taking the MSF DLL out of RESET.
8. Wait 1 ms.
9. Write 0x1490 into MSF_CLOCK_CONTROL to select the RCLK input, enable the RCLK DLL, and enable the RCLK_REF output clock.

10. If you are not using the flow control bus, go to step 16.
11. Write 0x14D0 into MSF_CLOCK_CONTROL to enable the TXCCLK output. The output clock is selected by MSF_TX_CONTROL[TXCCLK_SOURCE].
12. Start RXCCLK and wait for it to stabilize.
13. Write 0x34D0 into MSF_CLOCK_CONTROL to take the FC DLL out of reset. *Note:* the RXCCLK input clock must be present and stable before taking the FC DLL out of RESET.
14. Wait 1 ms.
15. Write 0x3CD0 into MSF_CLOCK_CONTROL to select the RXCCLK input and enable the RXCCLK DLL.
16. OR 0x32F into MSF_CLOCK_CONTROL (or an applicable subset). In this case, the receive and transmit calendars, the RSCLK output, and the MSF and Flow Control receive and transmit sections are all enabled; [Figure 3](#) ends with this step.
17. Train all interfaces as described in the appropriate section in Chapter 8 of the *Intel® IXP2800 Network Processor Hardware Reference Manual*.
18. Initialize CSIX_TYPEMAP, HWM_CONTROL, RX_PORTMAP, and RX_THREAD_FREELIST_TIMEOUT with appropriate values for your application.
19. If using CSIX, enable flow control as described in the appropriate section in Chapter 8 of the *Intel® IXP2800 Network Processor Hardware Reference Manual*.
20. OR in the appropriate enables in the upper four bits of MSF_RX_CONTROL and MSF_TX_CONTROL, to enable the SPI4/CSIX receive and transmit operations.
21. Clear all MSF interrupts by writing 0xFFFFFFFF to MSF_INTERRUPT_STATUS.
22. Enable appropriate MSF interrupts by initializing MSF_INTERRUPT_ENABLE.
23. Enable microengine receive threads and begin frame processing.

5.8 Media and Switch Fabric Interface(MSF) - IXP2400

5.8.1 IXP2400 MSF Address Map

Table 5-55. IXP2400 MSF Address Map

Register	Offset	Notes	Section
MSF_Rx_Control	0x0000		Section 5.8.2
MSF_Tx_Control	0x0004		Section 5.8.3
MSF_Interrupt_Status	0x0008		Section 5.8.4
MSF_Interrupt_Enable	0x000C		Section 5.8.5
CSIX_Type_Map	0x0010		Section 5.8.6
FC_Egress_Status	0x0014		Section 5.8.7
FC_Ingress_Status	0x0018		Section 5.8.8
reserved	0x001C		
reserved	0x0020		
HWM_Control	0x0024		Section 5.8.9
SRB_Override	0x0028		Section 5.8.10
reserved	0x002C		
Rx_Thread_Freelist_0	0x0030		Section 5.8.11
Rx_Thread_Freelist_1	0x0034		Section 5.8.11
Rx_Thread_Freelist_2	0x0038		Section 5.8.11
Rx_Thread_Freelist_3	0x003C	not used in IXP2800	Section 5.8.11
reserved	0x0040	Rx_Port_Map in IXP2800	
RBUF_Element_Done	0x0044		Section 5.8.12
Rx_MPHY_Poll_Limit	0x0048	Rx_Calendar_Length in IXP2800	Section 5.8.13
FCEFIFO_Validate	0x004C		Section 5.8.14
Rx_Thread_Freelist_Timeout_0	0x0050		Section 5.8.15
Rx_Thread_Freelist_Timeout_1	0x0054		Section 5.8.15
Rx_Thread_Freelist_Timeout_2	0x0058		Section 5.8.15
Rx_Thread_Freelist_Timeout_3	0x005C	not used in IXP2800	Section 5.8.15
Tx_Sequence_0	0x0060		Section 5.8.16
Tx_Sequence_1	0x0064		Section 5.8.16
Tx_Sequence_2	0x0068		Section 5.8.16
Tx_Sequence_3	0x006C	not used in IXP2800	Section 5.8.16
Tx_MPHY_Poll_Limit	0x0070	Tx_Calendar_Length in IXP2800	Section 5.8.17
Tx_MPHY_Status	0x0074	Rx_Pin_Deskew_0 in IXP2800	Section 5.8.18
Tx_MPHY_Status_Extension (Reserved in Rev A)	0x0078	Rx_Pin_Deskew_1 in IXP2800	Section 5.8.19



Register	Offset	Notes	Section
reserved	0x007C	Rx_Pin_Deskew_2 in IXP2800	
Rx_UP_Control_0	0x0080	not used in IXP2800	Section 5.8.20
Rx_UP_Control_1	0x0084	not used in IXP2800	Section 5.8.20
Rx_UP_Control_2	0x0088	not used in IXP2800	Section 5.8.20
Rx_UP_Control_3	0x008C	not used in IXP2800	Section 5.8.20
Tx_UP_Control_0	0x0090	not used in IXP2800	Section 5.8.21
Tx_UP_Control_1	0x0094	not used in IXP2800	Section 5.8.21
Tx_UP_Control_2	0x0098	not used in IXP2800	Section 5.8.21
Tx_UP_Control_3	0x009C	not used in IXP2800	Section 5.8.21
Rx_FIFO_Control_0	0x00A0		Section 5.8.22
Rx_FIFO_Control_1	0x00A4		Section 5.8.22
Rx_FIFO_Control_2	0x00A8		Section 5.8.22
Rx_FIFO_Control_3	0x00AC		Section 5.8.22
reserved	0x00B0 - 0x00EC		
MSF_Rx_RCOMP_Status	0x00F0	Rx IO buffers RCOMP Status	Section 5.8.23
MSF_Tx_RCOMP_Status	0x00F4	Tx IO buffers RCOMP Status	Section 5.8.24
MSF_Rx_RCOMP_Override	0x00F8	Used to override the drive settings of the Rx IO buffers.	Section 5.8.25
MSF_Tx_RCOMP_Override	0x00FC	Used to override the drive settings of the Tx IO buffers.	Section 5.8.26
FCIFIFO	0x0100 - 0x013C	FCIFIFO has 16 addresses for burst access. Burst must start at the lowest address.	Section 5.8.27
FCEFIFO	0x0140 - 0x017C	FCEFIFO has 16 addresses for burst access. Burst must start at the lowest address.	Section 5.8.28
reserved	0x1000 - 0x13FC	Tx_Calendar_# in IXP2800	
reserved	0x1400 - 0x17FC	Tx_Port_Status_# in IXP2800	
TBUF_ELEMENT_CONTROL_# (# range is 0 to 127, 0 to 63, or 0 to 31, for TBUF element size of 64, 128, or 256 bytes, respectively)	0x1800– 0x1BFC	Write TBUF_ELEMENT_CONTROL_# and set Element Valid. If done as 2 separate 32-bit writes, the write to the upper half of the register sets Element Valid.	Section 5.8.29
reserved (used to be TBUF_Element_Control_NV_#)	0x1C00 - 0x1FFC		
RBUF/TBUF	0x2000 - 0x3FFC	read = RBUF, write = TBUF	



5.8.2 MSF_Rx_Control

This control register defines a number of receive configuration parameters.

For IXP2400, the user may not write the RX_ENABLE bit at the same time as the remaining bits in the register. In other words, the first time the registers are written, the RX_ENABLE bit must be zero, to give the RX logic time to configure itself. The second time the registers are written, the interface may be enabled. (Note: This does not apply to IXP2800, where RX interfaces can be configured and enabled at the same time.)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
RX_EN[3:0]			RESERVED										RX_MODE	RX_WIDTH	RX_MPHY_EN	RX_MPHY_MODE	RX_MPHY_POLL_MODE	TX_CBUS_MODE	RESERVED	TX_CBUS_WIDTH	RESERVED	RESERVED	RESERVED	CSIX_FREELIST	RESERVED						RBUF_ELEMENT_SIZE	RESERVED

Bits	Field	Description	RW	Reset
[31:28]	Rx_En[3:0]	<p>Receive Enable.</p> <p>0—Receive section ignores the rx pins.</p> <p>1—Receive section enabled to receive transfers as defined.</p> <p>The mapping is as follows:</p> <p>[0] - channel 0 (SPHY), all channels (MPHY-4/-16 for Rev A and MPHY-4/-32 for Rev B), CSIX</p> <p>[1] - channel 1 (SPHY, MPHY-4)</p> <p>[2] - channel 2 (SPHY, MPHY-4)</p> <p>[3] - channel 3 (SPHY, MPHY-4)</p> <p>The RX_En bits should be set to 1 only after all the other RX configuration bits have been properly set, including other bits in this register. Set the other bits first, then rewrite this register with the appropriate RX_En bits set. These bits are not meant to be used dynamically. Once set, these bits should be left set. Moreover, the setting of these bits must be consistent with the mode and configuration of the MSF Receive interface that the user has set up. Otherwise, undefined behavior may result.</p> <p>For MPHY-4 mode, if a channel is not enabled but the slave device sends data to this channel, the IXP2400 will accept the data. This is a usage or slave device error, and will result in wasted RBUF space.</p>	RW	0

Bits	Field	Description	RW	Reset
[27:23]	reserved		RO	0
[22]	Rx_Mode	Controls the use of receive bus pins. 0: UTOPIA/POS Mode 1: CSIX Mode	RW	0
[21:20]	Rx_Width	Used to control use of the receive pins. Applicable only when running in UTOPIA/POS mode. 00: 1x32 01: 2x16 10: 4x8 11: 1x16_2x8	RW	0
[19]	Rx_MPHY_En	Used to enable MPHY operation on the receive interface. 0: single PHY mode 1: multi PHY mode	RW	0
[18]	Rx_MPHY_Mode	Used to select MPHY mode on the receive interface. This is ignored for POS-PHY Level 3 MPHY mode. 0: MPHY-4; a maximum of four ports. 1: MPHY-32; a maximum of thirty-two ports (For Rev A, MPHY-16; a maximum of sixteen ports). For Rev B MPHY-32 setting, the maximum number of ports for UTOPIA/POS Level 3 is 32 and for UTOPIA/POS Level 2 is 31.	RW	0
[17]	Rx_MPHY_Poll_Mode	Used to select FIFO status polling method on the receive interface in MPHY mode. From functionality perspective this bit is ignored for POS-PHY Level 3 MPHY-4 and MPHY-32 modes. Nevertheless, for IXP2400 rev B, this bit must be set to 0 for POS-PHY MPHY-4 configuration, due to implementation specific reason. 0: Direct status; FIFO status is carried on the RXFA[3:0] pins. May only be used in MPHY-4 mode. 1: Polled status; FIFO status must be polled and is carried on the RXPFA pin. May be used in either MPHY-4 or MPHY-32 (MPHY-16 in Rev A) modes.	RW	0
[16]	Tx_CBus_Mode	Determines operating mode for CBus transmit logic. • 0 - Simplex mode. • 1 - Full duplex mode. Simplex mode: • ME or Intel XScale® core writes to FCEFIFO address. • TXCSR_B output pin not used. • TXCFC input pin ignored. • TM_CRDY and TM_DRDY. Full Duplex mode: • CFrames routed from pins into FCEFIFO. • TXCSR_B output pin used to send TM_xRDY and SF_xRDY bits to ingress processor. • TXCFC input pin used to flow control CBus transmission.	RW	1
[15]	reserved		RO	0

Bits	Field	Description	RW	Reset
[14]	Rx_MPHY_Level2	When MSF is configured for UTOPIA or POS MPHY operation, this bit is used to select between Level 2 or Level 3 operation. 0: Level 3 operation 1: Level 2 operation This bit is Reserved in Rev A.	RW	0
[13]	Tx_Cbus_Width	0 --> 4-bit mode 1 --> 8-bit mode.	RW	0
[12:10]	reserved		RO	0
[9]	CSIX_Freelist	Determines how CFrames are mapped to Rx_Thread_Freelists 0: Data and Control CFrames go to different Rx_Thread_Freelists. 1: Data and Control CFrames go to the same Rx_Thread_Freelist.	RW	0
[8:4]	reserved		RO	0
[3:2]	RBUF_Element_Size	Indicates element size for RBUF. When MSF is configured for UTOPIA or POS, all elements have the same size. In CSIX mode with RBUF being divided into two partitions, both partitions have the same size elements. 00–64 bytes each 01–128 bytes each 10–256 bytes each 11–Reserved Note that in POS-PHY L3 MPHY configurations, the PHY device must send data in a burst size that matches with the configured RBUF_Element_Size. The only exception is when end of packet is reached. In this case, the burst size can be less than the RBUF_Element_Size.	RW	0
[1:0]	reserved		RO	0

5.8.3 MSF_Tx_Control

This control register defines a number of transmit configuration parameters. [Table 5-56](#) summarizes [Table 5-56](#) summarizes all the allowable major bus modes based on the contents of the MSF_Rx_Control/MSF_Tx_Control register. There are eight distinct major bus modes. Programming the register values for any illegal mode will result in undefined behavior.

For IXP2400, the user may not write the TX_ENABLE bit at the same time as the remaining bits in the register. In other words, the first time the registers are written, the TX_ENABLE bit must be zero, to give the TX logic time to configure itself. The second time the registers are written, the interface may be enabled. (Note: This does not apply to IXP2800, where TX interfaces can be configured and enabled at the same time.)



3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
TX_EN[3:0]				TX_FLUSH[3:0]				RESERVED	TX_MODE	TX_WIDTH	TX_MPHY_EN	TX_MPHY_MODE	TX_MPHY_POLL_MODE	RX_CBUS_MODE	RESERVED	TX_WIDTH	TX_WIDTH	TX_WIDTH	TX_WIDTH	RESERVED								TBUF_ELEMENT_SIZE	RESERVED

Bits	Field	Description	RW	Reset
[31:28]	Tx_En[3:0]	<p>Transmit Enable.</p> <p>0—Transmit section disabled, tx pins are static.</p> <p>1—Transmit section transmits TBUF elements as defined.</p> <p>The bits are mapped as follows:</p> <p>[0] - TBUF segment 0</p> <p>[1] - TBUF segment 1</p> <p>[2] - TBUF segment 2</p> <p>[3] - TBUF segment 3</p> <p>The TX_En bits should be set to 1 only after all the other TX configuration bits have been properly set, including other bits in this register. Set the other bits first, then rewrite this register with the appropriate TX_En bits set.</p> <p>These bits are not meant to be used dynamically. Once set, these bits should be left set. Moreover, the setting of these bits must be consistent with the mode and configuration of the MSF Transmit interface that the user has set up. Otherwise, undefined behavior may result.</p>	RW	0x0
[27:24]	Tx_Flush[3:0]	<p>These bits can be written to flush valid entries from TBUF. When a 1 is written to these bits, all valid bits of corresponding TBUF elements are cleared, and the internal element pointers used by that section of TBUF are reset.</p> <p>Tx_Flush[3] for TBUF partition 3</p> <p>Tx_Flush[2] for TBUF partition 2</p> <p>Tx_Flush[1] for TBUF partition 1</p> <p>Tx_Flush[0] for TBUF partition 0</p>	WO	0
[23]	reserved		RO	0
[22]	Tx_Mode	<p>Controls the use of transmit bus pins.</p> <p>0: UTOPIA/POS Mode</p> <p>1: CSIX Mode</p>	RW	0

Bits	Field	Description	RW	Reset
[21:20]	Tx_Width	Used to control use of transmit pins. Applicable only when running in UTOPIA/POS mode. 00: 1x32 01: 2x16 10: 4x8 11: 1x16_2x8	RW	0
[19]	Tx_MPHY_En	Used to enable MPHY operation on the transmit interface. 0: single PHY mode 1: multi PHY mode	RW	0
[18]	Tx_MPHY_Mode	Used to select MPHY mode on the transmit interface. 0: MPHY-4; a maximum of four ports. 1: MPHY-32; a maximum of thirty-two ports (For Rev A, MPHY-16; a maximum of sixteen ports). For Rev B MPHY-32 setting, the maximum number of ports for UTOPIA/POS Level 3 is 32 and for UTOPIA/POS Level 2 is 31.	RW	0
[17]	Tx_MPHY_Poll_Mode	Used to select FIFO status polling method on the transmit interface in MPHY mode. 0: Direct status indication; FIFO status is carried on the TXFA[3:0] pins. May only be used in MPHY-4 mode. 1: Polled status; FIFO status must be polled and is carried on the TXPFA pin. May be used in either MPHY-4 or MPHY-32 (MPHY-16 in Rev A) modes.	RW	0
[16]	Rx_CBus_Mode	Determines operating mode for CBus receive logic. • 0 - Simplex mode. • 1 - Full duplex mode. Simplex mode: • CFrames come directly from switch fabric. • RXCSRB input pin ignored. • RXCFC output pin not used. • SF_CRDY and SF_DRDY extracted from incoming CFrames. Full Duplex mode: • CFrames come from egress processor. • RXCSRB input pin used to receive SF_xRDY and TM_xRDY from egress processor. • RXCFC output pin used to flow control CBus transmission.	RW	1
[15]	reserved		RO	0
[14]	Tx_MPHY_Level2	When MSF is configured for UTOPIA or POS MPHY operation, this bit is used to select between Level 2 or Level 3 operation. 0: Level 3 operation 1: Level 2 operation This bit is Reserved in Rev A.	RW	0
[13]	Rx_Cbus_Width	0 --> 4-bit mode 1 --> 8-bit mode	RW	0
[12:4]	reserved		RO	0

Bits	Field	Description	RW	Reset
[3:2]	TBUF_Element_Size	Indicates element size for TBUF. When MSF is configured for UTOPIA or POS, all elements have the same size. In CSIX mode with TBUF being divided into two partitions, both partitions have the same size elements. 00–64 bytes each 01–128 bytes each 10–256 bytes each 11–Reserved	RW	0
[1:0]	reserved		RO	0

Table 5-56. IXP2400 MSF Allowable Major Bus Modes

{Rx,Tx}_Mode	{Rx,Tx}_Width[1:0]	{Rx,Tx}_MPHY_En	{Rx,Tx}_MPHY_Mode	{Rx,Tx}_MPHY_Poll_Mode	Operating Mode	Notes
0	00	0	X	X	U/P, 1x32, SPHY	Rx_UP_Control_0 and Tx_UP_Control_0 are used to configure port behavior.
0	00	1	0	0	U/P, 1x32, MPHY-4, direct status	Rx_UP_Control_0 and Tx_UP_Control_0 are used to configure port behavior.
0	00	1	0	1	U/P, 1x32, MPHY-4, polled status	Rx_UP_Control_0 and Tx_UP_Control_0 are used to configure port behavior.
0	00	1	1	0	illegal	Direct status not allowed for MPHY-32 (MPHY-16 in Rev A) mode.
0	00	1	1	1	U/P, 1x32, MPHY-32 (MPHY-16 in Rev A), polled status	Rx_UP_Control_0 and Tx_UP_Control_0 are used to configure port behavior.
0	01	0	X	X	U/P, 2x16, SPHY	Rx_UP_Control_{0,2} and Tx_UP_Control_{0,2} are used to configure port behavior.
0	01	1	0	0	illegal	
0	01	1	0	1	illegal	
0	01	1	1	0	illegal	
0	01	1	1	1	U/P, 1x16 MPHY-32 polled status and 1x16 SPHY	X16 MPHY mode illegal in Rev A
0	10	0	X	X	U/P, 4x8, SPHY	Rx_UP_Control_{0,1,2,3} and Tx_UP_Control_{0,1,2,3} are used to configure port behavior.
0	10	1	X	X	illegal	X8 MPHY mode not supported.
0	11	0	X	X	U/P, 1x16+2x8, SPHY	Rx_UP_Control_{0,2,3} and Tx_UP_Control_{0,2,3} are used to configure port behavior.
0	11	1	0	0	illegal	

Table 5-56. IXP2400 MSF Allowable Major Bus Modes (Continued)

{Rx,Tx}_Mode	{Rx,Tx}_Width[1:0]	{Rx,Tx}_MPHY_En	{Rx,Tx}_MPHY_Mode	{Rx,Tx}_MPHY_Poll_Mode	Operating Mode	Notes
0	11	1	0	1	illegal	
0	11	1	1	0	illegal	
0	11	1	1	1	U/P, 1x16 MPHY-32 polled status and 2x8 SPHY	X16 MPHY mode illegal in Rev A
1	00	X	X	X	CSIX, 1x32	
1	01	X	X	X	illegal	CSIX mode only supported for 1x32.
1	10	X	X	X	illegal	CSIX mode only supported for 1x32.
1	11	X	X	X	illegal	CSIX mode only supported for 1x32.

5.8.4 MSF_Interrupt_Status

This register holds error status. When any of these bits is set, MSF generates an interrupt to the Intel XScale® core.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0													
RESERVED																FCEFIFO_OVERFLOW										RBUF_OVERFLOW_COUNT										RESERVED	FCIFIFO_ERROR	RESERVED	TBUF_ERROR	RESERVED	RESERVED	VP_ERROR	HP_ERROR

Bits	Field	Description	RW	Reset
[31:24]	reserved		RO	0
[23:16]	FCEFIFO_Overflow	<p>Full Duplex Mode: CSIX CFrame mapped to FCEFIFO arrived and FCEFIFO did not have sufficient room. The entire CFrame was discarded. This field counts up by 1 for every discarded CFrame. FCEFIFO can hold up to 256 CWords.</p> <p>Simplex Mode: Software attempted to write data into FCEFIFO and it is completely full. (FCEFIFO can hold up to 255 entries in this mode). This field counts up by 1 for every discarded CWord.</p> <p>This field saturates at 0xFF. In other words, when the count reaches 0xFF, it will stop counting, so as to not roll over to zero.</p> <p>Note that these bits need to be cleared all at the same time by writing 0xFF to them.</p>	RW1C	0
[15:8]	RBUF_Overflow_Count	<p>Data was received on Rx pins successfully but no RBUF element was free to accept it. The data may get discarded, if more data arrives at the Rx pins but not enough RBUF entries get freed up. This field counts up by 1 for every mpacket that is received while no free RBUF element is available. This incrementing field saturates at 0xFC. In other words, when the count reaches 0xFC, it will stop incrementing. Note that the IXP2400 and IXP2800 behavior for this field differs.</p> <p>Note that these bits need to be cleared all at the same time by writing 0xFF to them.</p>	RW1C	0
[7]	reserved		RO	0
[6]	FCIFIFO_Error	<p>FCIFIFO CFrame was discarded due to either:</p> <ul style="list-style-type: none"> • Horizontal parity error • Vertical parity error • Premature RXCSOF (before entire payload length was received) • Overflow - CFrame with Payload Size greater than space available in FCIFIFO. Once the overflow condition occurs, subsequent Cframes get dropped until more space in FCIFIFO becomes available and the overflow condition disappears. Only one interrupt is sent during the entire duration of the overflow condition, regardless of the number of dropped CFrames due to the overflow condition. 	RW1C	0
[5]	reserved		RO	0
[4]	TBUF_Error	<p>Transmit Control Word programming error. Set by either:</p> <ul style="list-style-type: none"> • Prepend Length + Payload Length > TBUF element size. Total number of bytes sent is truncated to element size. • Prepend Length + Payload Length not an integer multiple of bus width and the EOP bit is not set in the Transmit Control Word. A full mpacket's worth of data will be transmitted, regardless of the value of Prepend Length + Payload Length. 	RW1C	0

Bits	Field	Description	RW	Reset
[3:2]	reserved		RO	0
[1]	VP_Error	Incorrect Vertical Parity on a received CFrame in CSIX mode.	RW1C	0
[0]	HP_Error	Incorrect Horizontal Parity received in UTOPIA, POS-PHY, or CSIX modes.	RW1C	0

5.8.5 MSF_Interrupt_Enable

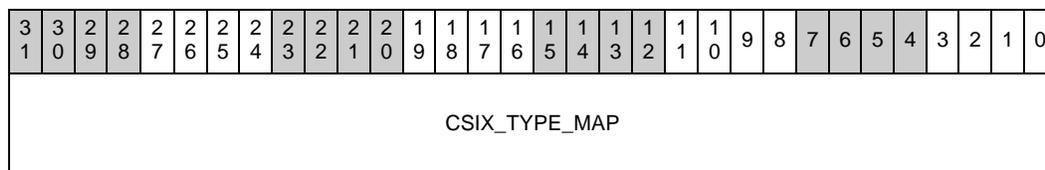
This register holds enable bits for individual error types. This register is ANDed with MSF_Interrupt_Status. If the result is not zero MSF generates an interrupt signal to the Intel XScale® core.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0							
RESERVED														FCE_FIFO_OVERFLOW	RESERVED										RBUF_OVERFLOW_COUNT	RESERVED	FCIFIFO_ERROR	RESERVED	TBUF_ERROR	RESERVED	RESERVED	VP_ERROR	HP_ERROR

Bits	Field	Description	RW	Reset
[31:17]	reserved		RO	0
[16]	FCE_FIFO_Overflow		RW	0
[15:9]	reserved		RO	0
[8]	RBUF_Overflow_Count		RW	0
[7]	reserved		RO	0
[6]	FCIFIFO_Error		RW	0
[5]	reserved		RO	0
[4]	TBUF_Error		RW	0
[3:2]	reserved		RO	0
[1]	VP_Error		RW	0
[0]	HP_Error		RW	0

5.8.6 CSIX_Type_Map

This register is determines how CSIX CFrames are mapped to RBUF elements. This register must not be changed while CSIX CFrames are being received.

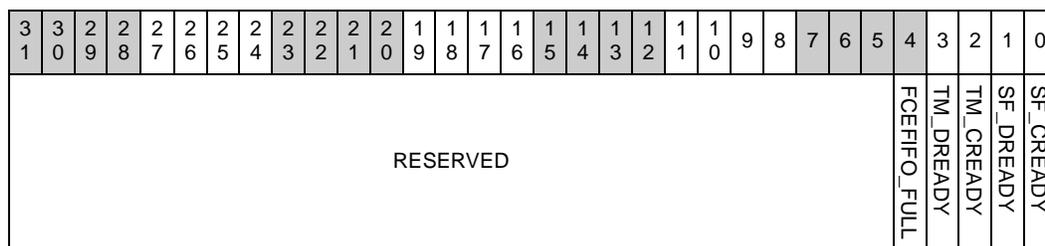


Bits	Field	Description	RW	Reset
[31:0]	CSIX_Type_Map	These pairs of bits map CFrame Type field to either RBUF or FCEFIFO as show. Bits [1:0] is for type 0x0, bits [3:2] is for type 0x1, etc. <ul style="list-style-type: none"> • 00 - Discard • 01 - RBUF Control • 10 - RBUF Data • 11 - FCEFIFO 	RW	0

5.8.7 FC_Egress_Status

This register holds the link level flow control information received from the Switch Fabric, and the status of RBUF and FCEFIFO. In Full Duplex mode, this information is transmitted out of the egress processor on TXCSRB.

Note: The actual update into bits [1:0] of this register is only done when the CFrame is received error-free. If there is an error on a CFrame then those two bits are cleared. Errors will set bits as defined in MSF_Interrupt_Status.

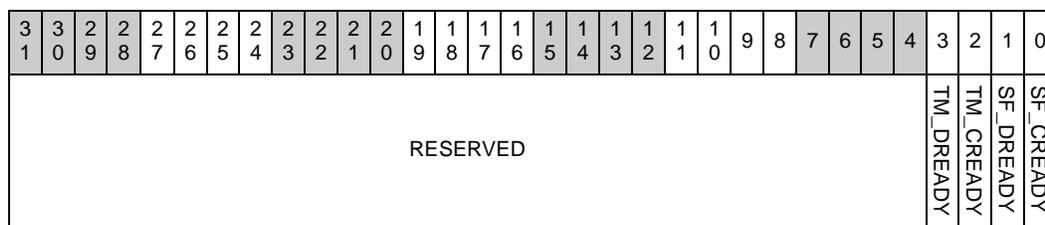


Bits	Field	Description	RW	Reset
[31:5]	reserved		RO	0
[4]	FCEFIFO_Full	Indicates in FCEFIFO is full or not, based on HWM_Control[FCEFIFO_HWM]. Note that this bit is only applicable in Simplex mode.	RO	0

Bits	Field	Description	RW	Reset
[3]	TM_DReady	Value for ingress IXP2400 to use for DRDY in CFrames sent to the switch fabric. Deasserted when RBUF CSIX data partition is full (based on HWM_Control[RBUF_D_HWM]). Full Duplex mode: transmitted on TXCSR bit 8. Simplex mode: DRDY value to transmit (in bit 7 of byte 0) of CSIX base headers sent on TXCDAT.	RO	0
[2]	TM_CReady	Value for Ingress IXP2400 to use for CRDY in CFrames sent to Switch Fabric. In Full Duplex Mode, deasserted when RBUF CSIX Control Partition is full (based on HWM_Control[RBUF_C_HWM]) or when FCEFIFO is full (based on HWM_Control[FCEFIFO_HWM]). In Simplex Mode deasserted only when RBUF CSIX Control Partition is full. In Full Duplex Mode -- transmitted on TXCSR bit 7. In Simplex Mode -- CRDY value to transmit (in bit 6 of byte 0) in CSIX Base Headers sent on TXCDAT.	RO	0
[1]	SF_DReady	Data Ready receive in CSIX base headers (bit 7 of byte 0), updated after CFrame is received. Cleared if an error is detected on a received CFrame. Full Duplex mode: transmitted on TXCSR bit 6.	RO	0
[0]	SF_CReady	Control Ready receive in CSIX base headers (bit 6 of byte 0), updated after CFrame is received. Cleared if an error is detected on a received CFrame. Full Duplex mode: transmitted on TXCSR bit 5.	RO	0

5.8.8 FC_Ingress_Status

This register holds the link level flow control information received on RXCSR. It is used as ready information on CSIX base headers, and used to enable or disable transmission of CFrames to the switch fabric.



Bits	Field	Description	RW	Reset
[31:4]	reserved		RO	0
[3]	TM_DReady	Simplex mode: Always 0. Full Duplex mode: CRDY value to transmit (in bit 7 of byte 0) of CSIX base headers. This value is received on RXCSR bit 8.	RO	0

Bits	Field	Description	RW	Reset
[2]	TM_CReady	Simplex mode: The number of free CWords in FCIFIFO is below the high watermark as programmed in HWM_Control[FCIFIFO_Ext_HWM]. Full Duplex mode: CRDY value to transmit (in bit 6 of byte 0) of CSIX base headers. This value is received on RXCSRB bit 7.	RO	0
[1]	SF_DReady	Hardware control of transmission of CSIX data elements. • 0 - stop sending data CFrames to the switch fabric. • 1 - OK to send data CFrames to the switch fabric. Full Duplex mode: received on RXCSRB bit 6. Simplex mode: CRDY of CSIX base headers (bit 7 of byte 0) received on RXCDATA, updated as each CFrame is received. Cleared if an error is detected on a received CFrame.	RO	0
[0]	SF_CReady	Hardware control of transmission of CSIX control elements. • 0 - stop sending control CFrames to the switch fabric. • 1 - OK to send control CFrames to the switch fabric. Full Duplex mode: received on RXCSRB bit 5. Simplex mode: CRDY of CSIX base headers (bit 6 of byte 0) received on RXCDATA, updated as each CFrame is received. Cleared if an error is detected on a received CFrame.	RO	0

5.8.9 HWM_CONTROL

This register is used to control high watermarks for RBUF, FCEFIFO, and FCIFIFO. This register must not be changed while data is being received.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																				FCIFIFO_EXT_HWM	FCIFIFO_INT_HWM	FCEFIFO_HWM	RESERVED	RBUF_D_HWM	RBUF_C_HWM						

Bits	Field	Description	RW	Reset
[31:12]	reserved		RO	0

Bits	Field	Description	RW	Reset
[11:10]	FCIFIFO_Ext_HWM	Flow Control Ingress FIFO High Watermark for external use. Simplex mode: TM_CRDY to the switch fabric is deasserted if the number of CWord entries used in the FCIFIFO is > the programmed high watermark. Full duplex mode: RXFC is asserted if the number of CWord entries used in the FCIFIFO is > the programmed watermark. Note this is different than the IXP2800. 00: 32 CWord entries are used 01: 64 CWord entries are used 10: 128 CWord entries are used 11: 192 CWord entries are used	RW	0
[9:8]	FCIFIFO_Int_HWM	Flow Control Ingress FIFO High Watermark for internal use. FCI_Near_Full signal to the MEs is asserted if the number of CWords entries used in the FCIFIFO > the programmed watermark. Note this is different than the IXP2800. 00: 16 CWord entries are used 01: 32 CWord entries are used 10: 64 CWord entries are used 11: 128 CWord entries are used	RW	0
[7:6]	FCEFIFO_HWM	Flow Control Egress FIFO High Watermark. FC_Egress_Status[FCEFIFO_Full] is asserted if Simplex mode is configured and the number of CWord entries used in the FCEFIFO is > the programmed water mark. For Full-duplex configuration, the TM_CRDY bit of outgoing CFrames gets deasserted when the number of CWord entries used in the FCEFIFO is > the programmed water mark. For Full-duplex configuration: 00: 64 CWord entries are used 01: 128 CWord entries are used 10: 184 CWord entries are used 11: 216 CWord entries are used For Simplex configuration: 00: 64 CWord entries are used 01: 128 CWord entries are used 10: 192 CWord entries are used 11: 224 CWord entries are used	RW	0
[5:4]	reserved		RO	0

Bits	Field	Description	RW	Reset
[3:2]	RBUF_D_HWM	<p>RBUF Data High Watermark. If the number of RBUF entries used in the data partition is > the number in this field, then the TM_DRDY bit sent to switch fabric must be deasserted to flow control the switch fabric and prevent it from sending any more data CFrames. Note this is different than the IXP2800. The encoding is as follows:</p> <p>With 64B element size 00: 24 entries are used 01: 48 entries are used 10: 72 entries are used 11: 84 entries are used</p> <p>With 128B element size 00: 12 entries are used 01: 24 entries are used 10: 36 entries are used 11: 42 entries are used</p> <p>With 256B element size 00: 6 entries are used 01: 12 entries are used 10: 18 entries are used 11: 21 entries are used</p>	RW	0
[1:0]	RBUF_C_HWM	<p>RBUF Control High Watermark. If the number of RBUF entries used in the control partition is > the number in this field, then the TM_CRDY bit sent to switch fabric must be deasserted to flow control the switch fabric and prevent it from sending any more control CFrames. Note this is different than the IXP2800. The encoding is as follows:</p> <p>With 64B element size 00: 8 entries are used 01: 16 entries are used 10: 24 entries are used 11: 28 entries are used</p> <p>With 128B element size 00: 4 entries are used 01: 8 entries are used 10: 12 entries are used 11: 14 entries are used</p> <p>With 256B element size 00: 2 entries are used 01: 4 entries are used 10: 6 entries are used 11: 7 entries are used</p>	RW	0

5.8.10 SRB_Override

This register allows the TXCSR_B and/or RXCSR_B pin data to be overridden. These pins are used only in CBUS full duplex mode, not in simplex mode, so this register has no affect in simplex mode.

This Register performs its documented functionality in both simplex and full-duplex modes for the IXP2800. The IXP2400 only supports full-duplex mode.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																					TXCSR_B_FORCE_EN	RXCSR_B_FORCE_EN	TXCSR_B_FORCE	RXCSR_B_FORCE						

Bits	Field	Description	RW	Reset
[31:10]	reserved		RO	0
[9]	TXCSR_B_Force_En	TXCSR_B Enable. Used to tell hardware whether to use transmit serialized flow control information generated by hardware or to use the values in the TXCSR_B_Force field. This is valid only if MSF_Rx_Control[Receive_Mode] is CSIX. 0: Drive TXCSR_B output from FC_Egress_Status. 1: Drive TXCSR_B output from TXCSR_B_Force field.	RW	0
[8]	RXCSR_B_Force_En	RXCSR_B Enable. Used to tell hardware whether to use serialized flow control information received on the RXCSR_B input or to use the values in the RXCSR_B_Force field. This is valid only if MSF_Tx_Control[Transmit_Mode] is CSIX. 0: Use RXCSR_B input signal. 1: Use RXCSR_B_Force field.	RW	0
[7:4]	TXCSR_B_Force	This is used to provide hardware with software-chosen values to send on the TXCSR_B pins in place of the ones generated by internal hardware. 7: SF_CRDY 6: SF_DRDY 5: TM_CRDY 4: TM_DRDY	RW	0
[3:0]	RXCSR_B_Force	This is used to provide hardware with software-chosen values to use in place of the ones received on the RXCSR_B pins. 3: SF_CRDY 2: SF_DRDY 1: TM_CRDY 0: TM_DRDY	RW	0

Bits	Field	Description	RW	Reset
[3:0]	RXCSRB_Force	This is used to provide hardware with software-chosen values to use in place of the ones received on the RXCSRB pins. 3: SF_CRDY 2: SF_DRDY 1: TM_CRDY 0: TM_DRDY	RW	0

5.8.11 Rx_Thread_Freelist_{0.3}

MEs write to this register to add a Context to the Rx_Thread_Freelist. Note that the Microengine number is specified differently for the IXP2800.

Refer to the IXP2400 Hardware Reference Manual (HRM) for definition of the Element status.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED															SIGNAL_NUMBER			RESERVED		ME_NUMBER			THREAD		TRANSFER_REG				

Bits	Field	Description	RW	Reset
[31:16]	Reserved	Read returns 0	RO	0
[15:12]	Signal_Number	Which signal to deliver when pushing the Element status. Read returns 0.	WO	undef
[11:10]	reserved		RO	0
[9:7]	ME_Number	Indicates which ME will get the RBUF Element status pushed to it. The valid ME numbers for this field are 0 through 7. Values 0 to 3 correspond to MEs in cluster 0. Values 4 to 7 correspond to MEs in cluster 1. Read returns 0.	WO	undef
[6:4]	Thread	Indicates which thread will get the RBUF Element status pushed to it. Read returns 0.	WO	undef
[3:0]	Transfer Reg	Indicates which SRAM Transfer Registers will get the RBUF Element status pushed to it. Two consecutive registers, starting with the one in this field, are written. Read returns 0.	WO	undef

5.8.12 RBUF_Element_Done

Threads write to this address to free up an RBUF element so that it can be reused.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																								ELEMENT_TO_FREE							

Bits	Field	Description	RW	Reset
[6:0]	Element to Free	Indicates the number of the element to free up. Note: After reset, the RBUF Element Freelist (s) are empty. As part of the initialization code, the software must write RBUF numbers into the RBUF_Element_Done CSR to place RBUF entries into the freelist before enabling receive operation. Read returns 0. Note: Each element can only be freed once, until it is used again when new data is received. Freeing elements multiple times can result in undefined hardware behavior.	WO	undef
[31:7]	Reserved	Read returns 0.	RO	0

5.8.13 Rx_MPHY_Poll_Limit

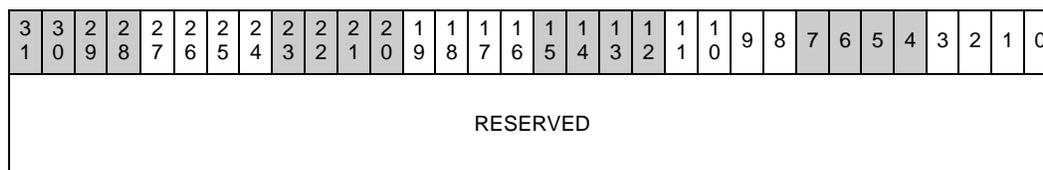
This register is used to tell hardware how many ports are present for MPHY-32 (MPHY-16 in Rev A) and MPHY-4 modes (with shared status). Note that for MPHY-4 modes, the reset value of 0x1F (0xF in Rev A) will be too big and the user must configure the Poll_Limit field with the proper value.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																								POLL_LIMIT							

Bits	Field	Description	RW	Reset
[4:0]	Poll_Limit	<p>This field is used to inform hardware the number of the highest port number. It is used to prevent hardware from polling non-existent ports.</p> <p>Note that the maximum number of ports have been increased to 32 in Rev B, from 16 in Rev A. For UTOPIA L2 and POS-PHY L2, the maximum number of ports is 31.</p> <p>For example, if there are four ports present, 0x3 should be written to this field; the polling sequence will then be 0x0, 0x1, 0x2, 0x3, 0x0, 0x1, ...</p> <p>If there are sixteen ports present, 0xf should be written to this field. The polling sequence will then be 0x0, 0x1, 0x2, 0x3, ..., 0xe, 0xf, 0x0, 0x1, ...</p> <p>If there are 31 ports present, assuming UTOPIA L2 or POS-PHY L2, 0x1e should be written to this field. The polling sequence will then be 0x0, 0x1f, 0x1, 0x1f, 0x2, ..., 0x1d, 0x1f, 0x1e, 0x1f, 0x0, 0x1f, ... where 0x1f is the null address which is automatically inserted by the hardware to enforce dead cycles on RXPFA/TXPFA as required by the UTOPIA L2 or POS-PHY L2 specifications.</p> <p>If there are 32 ports present, assuming UTOPIA L3 or POS-PHY L3, 0x1f should be written to this field. The polling sequence will then be 0x0, 0x1, 0x2, 0x3, ..., 0x1e, 0x1f, 0x0, ... In this case a null address is not needed because the connection between IXP2400 and the media device is assumed to be point to point and no dead cycles are ever needed.</p> <p>This implies that port numbers must always start at 0 and must always be contiguous.</p>	RW	0x1F (0xF in Rev A)
[31:5]	reserved		RO	0

5.8.14 FCEFIFO_Validate

This register is used to validate a CFrame written into FCEFIFO by software in simplex mode. The CFrame will not be transmitted on the TXCDATA pins until it is validated. That data is not used; any write to this register does the validate.



Bits	Field	Description	RW	Reset
[31:0]	reserved	Read returns 0.	WO	undef

5.8.15 Rx_Thread_Freelist_Timeout_{0..3}

There is one Rx_Thread_Freelist_Timeout register associated with each Rx_Thread_Freelist.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																TMEOUT_INTERVAL														

Bits	Field	Description	RW	Reset
[12:0]	Timeout_Interval	<p>This is the number of CPP bus clocks, starting from the autopush of a null or non-null Receive Status Word, that are allowed to elapse before triggering the autopush of a null Receive Status Word.</p> <p>Every time any Receive Status Word is autopushed the timer is reset and restarted. If no new receive traffic has come in and the timeout interval has been reached, hardware will automatically autopush a null Receive Status Word to the next thread in the Rx_Thread_Freelist.</p> <p>A value of 0x0000 means that the timer is disabled.</p> <p>When enabled, the Timeout_Interval value should be set to be high enough so that the rate of timeout is slower than the rate of new elements being added to the corresponding Rx_Thread_Freelist. Otherwise, new elements to the Rx_Thread_Freelist may continue to encounter timeout and receive null Receive Status Word.</p>	RW	0x0000
[31:13]	Reserved		RO	0

5.8.16 Tx_Sequence_{0..3}

A read of this register provides a wrapping count of the number of TBUF elements that have been transmitted from the partition. The number of partitions in used is based on MSF_Tx_Control[Tx_Mode] and MSF_Tx_Control[Tx_Width]. The count advances when the entire content of the element has been transmitted (i.e. so it is safe to write new data into the element).

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
E M P T Y	RESERVED																SEQUENCE													

Bits	Field	Description	RW	Reset
[31]	Empty	Indicates if there are any valid elements for this partition. 0: One or more elements are valid. 1: No elements are valid.	RO	1
[30:8]	reserved		RO	0
[7:0]	Sequence	Sequence count of elements transmitted for this partition. The count always counts from 0 to 255 and then wraps back to 0, regardless of the number of elements in the partition.	RO	0

5.8.17 Tx_MPHY_Poll_Limit

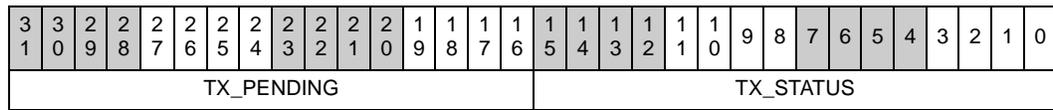
This register is used to tell hardware how many ports are present for MPHY-32 (MPHY-16 in Rev A) and MPHY-4 modes (with shared status). Note that for MPHY-4 modes, the reset value of 0xF will be too big and the user must configure the Poll_Limit field with the proper value.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																												POLL_LIMIT			

Bits	Field	Description	RW	Reset
[31:5]	reserved		RO	0
[4:0]	Poll_Limit	<p>This field is used to inform hardware the number of the highest port number. It is used to prevent hardware from polling non-existent ports.</p> <p>Note that the maximum number of ports have been increased to 32 in Rev B, from 16 in Rev A. For UTOPIA L2 and POS-PHY L2, the maximum number of ports is 31.</p> <p>For example, if there are four ports present, 0x3 should be written to this field; the polling sequence will then be 0x0, 0x1, 0x2, 0x3, 0x0, 0x1, ...</p> <p>If there are sixteen ports present, 0xf should be written to this field. The polling sequence will then be 0x0, 0x1, 0x2, 0x3, ..., 0xe, 0xf, 0x0, 0x1, ...</p> <p>If there are 31 ports present, assuming UTOPIA L2 or POS-PHY L2, 0x1e should be written to this field. The polling sequence will then be 0x0, 0x1f, 0x1, 0x1f, 0x2, ..., 0x1d, 0x1f, 0x1e, 0x1f, 0x0, 0x1f, ... where 0x1f is the null address which is automatically inserted by the hardware to enforce dead cycles on RXPFA/TXPFA as required by the UTOPIA L2 or POS-PHY L2 specifications.</p> <p>If there are 32 ports present, assuming UTOPIA L3 or POS-PHY L3, 0x1f should be written to this field. The polling sequence will then be 0x0, 0x1, 0x2, 0x3, ..., 0x1e, 0x1f, 0x0, ... In this case a null address is not needed because the connection between IXP2400 and the media device is assumed to be point to point and no dead cycles are ever needed.</p> <p>This implies that port numbers must always start at 0 and must always be contiguous.</p>	RW	0x1F (0xF in Rev A)

5.8.18 Tx_MPHY_Status

This register is roughly analogous to the Transmit Calendar Queue in IXP2800. It is intended to be used only in MPHY-16 (Rev A) and MPHY-32 (Rev B) modes to aid in transmit scheduling. In these MPHY modes, TBUF functions as a single FIFO; all transmit traffic for all the MPHY ports funnels into this single FIFO. In order to prevent head-of-line blocking, software must use the Tx_MPHY_Status register to determine when it is “safe” to push traffic for a given port into TBUF. The status of each transmit FIFO in the PHY is visible to software in the Tx_MPHY_Status register, as well as other hints; software reads the Tx_MPHY_Status register to determine which ports can accept traffic, then only sends transmit data to those ports.



Bits	Field	Description	RW	Reset
[31:16]	Tx_Pending	UTOPIA Mode: [0] corresponds to port 0, ..., [15] corresponds to port 15. This bit is set when a cell is pushed into TBUF; it is cleared automatically by hardware after the assertion of TXSOF on the pins. 0: TBUF does not contain any cell in flight to the given port. 1: TBUF contains a cell in flight. POS-PHY Mode: [0] corresponds to port 0, ..., [15] corresponds to port 15. Hardware maintains a counter for each port; every time an mpacket for the port is pushed into TBUF, the counter is incremented; every time the mpacket is drained from TBUF and sent to the PHY the counter is decremented. The Tx_Pending bit is asserted whenever the counter is non-zero. 0: TBUF contains does not contain any mpackets destined for this port. 1: TBUF contains one or more mpackets destined for this port.	RO	0

Bits	Field	Description	RW	Reset
[15:0]	Tx_Status	<p>UTOPIA mode: [0] corresponds to port 0, ..., [15] corresponds to port 15. 0 = PHY's TX FIFO is full 1 = PHY's TX FIFO is not full and can accept at least one more cell.</p> <p>The TX thread can only send one cell to each port whose Tx_Status flag is set. When a transmit control word is written, thereby initiating the transmission of a cell to a certain port, the Tx_Pending flag for the given port is set; this indicates that a cell transmit is in progress for that given port and that Tx_Status is now stale; when hardware sends the cell out on the TX pins, it will update the Tx_Status bit for that port with the most current port status before clearing the Tx_Pending flag.</p> <p>POS-PHY mode: [0] corresponds to port 0, ..., [15] corresponds to port 15. 0 = PHY's TX FIFO is near full 1 = PHY's TX FIFO is not full and can accept at least "n" more mpackets of data.</p> <p>This bit corresponds to the TX FIFO status flag in the PHY. The PHY's TX FIFO has a programmable threshold which software initializes to the desired value. Software knows what the threshold is, and that if the PHY's TX FIFO contains less data than this, it can accept at least "n" mpackets worth of data before overflowing, where "n" is the difference between the FIFO size and the threshold.</p> <p>After the transmit logic has been enabled, the polling FSM will update the contents of this field. The TX thread can send up to "n" mpackets to each port whose Tx_Status bit is "1" This will cause the Tx_Pending flag to be set, indicating that Tx_Status is stale. When all mpackets for a given port have been transmitted hardware will update the Tx_Status bit for that port with the most up to date state before letting the Tx_Pending bit transition from 1 to 0.</p> <p><i>Hardware will never drop transmit data when transmitting data to a port whose Tx_Status flag is 0 due to the presence of the TXSFA signal; however sub-optimal performance may result due to head-of-line blocking and stalling.</i></p>	RO	0

5.8.19 Tx_MPHY_Status_Extension

This register serves the same function as Tx_MPHY_Status, except that it covers the 16 ports that are added in Rev B.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	
TX_PENDING																TX_STATUS															

Bits	Field	Description	RW	Reset
[31:16]	Tx_Pending	<p>UTOPIA Mode: [0] corresponds to port 16, ..., [15] corresponds to port 31. This bit is set when a cell is pushed into TBUF; it is cleared automatically by hardware after the assertion of TXSOF on the pins. 0: TBUF does not contain any cell in flight to the given port. 1: TBUF contains a cell in flight.</p> <p>POS-PHY Mode: [0] corresponds to port 16, ..., [15] corresponds to port 31. Hardware maintains a counter for each port; every time an mpacket for the port is pushed into TBUF, the counter is incremented; every time the mpacket is drained from TBUF and sent to the PHY the counter is decremented. The Tx_Pending bit is asserted whenever the counter is non-zero. 0: TBUF contains does not contain any mpackets destined for this port. 1: TBUF contains one or more mpackets destined for this port.</p>	RO	0



Bits	Field	Description	RW	Reset
[15:0]	Tx_Status	<p>UTOPIA mode: [0] corresponds to port 16, ..., [15] corresponds to port 31. 0 = PHY's TX FIFO is full 1 = PHY's TX FIFO is not full and can accept at least one more cell.</p> <p>The TX thread can only send one cell to each port whose Tx_Status flag is set. When a transmit control word is written, thereby initiating the transmission of a cell to a certain port, the Tx_Pending flag for the given port is set; this indicates that a cell transmit is in progress for that given port and that Tx_Status is now stale; when hardware sends the cell out on the TX pins, it will update the Tx_Status bit for that port with the most current port status before clearing the Tx_Pending flag.</p> <p>POS-PHY mode: [0] corresponds to port 16, ..., [15] corresponds to port 31. 0 = PHY's TX FIFO is near full 1 = PHY's TX FIFO is not full and can accept at least "n" more mpackets of data.</p> <p>This bit corresponds to the TX FIFO status flag in the PHY. The PHY's TX FIFO has a programmable threshold which software initializes to the desired value. Software knows what the threshold is, and that if the PHY's TX FIFO contains less data than this, it can accept at least "n" mpackets worth of data before overflowing, where "n" is the difference between the FIFO size and the threshold.</p> <p>After the transmit logic has been enabled, the polling FSM will update the contents of this field. The TX thread can send up to "n" mpackets to each port whose Tx_Status bit is "1" This will cause the Tx_Pending flag to be set, indicating that Tx_Status is stale. When all mpackets for a given port have been transmitted hardware will update the Tx_Status bit for that port with the most up to date state before letting the Tx_Pending bit transition from 1 to 0.</p> <p><i>Hardware will never drop transmit data when transmitting data to a port whose Tx_Status flag is 0 due to the presence of the TXSFA signal; however sub-optimal performance may result due to head-of-line blocking and stalling.</i></p>	RO	0

5.8.20 Rx_UP_Control_{0..3}

There are four of these registers, one per port. The contents of these registers apply only if the chip is configured for UTOPIA/POS mode.

For MPHY4 mode configurations, all 4 registers, Rx_UP_Control_{0..3}, must be programmed with the same value. For POS-PHY Level 3 or UTOPIA Level 3 MPHY-32 (MPHY-16 in Rev A) mode configurations, the content in Rx_UP_Control_0 register applies to all the ports, and the content of the Rx_UP_Control_{1..3} registers are ignored. The whole 32-bit interface is utilized in these mode configurations. For POS-PHY Level 2 or UTOPIA Level 2 MPHY-32 (only available in Rev B) mode configurations, the content in Rx_UP_Control_0 register applies to all the MPHY ports, and the content of the Rx_UP_Control_1 register is ignored. In these mode configurations, only 16 bit out of the 32-bit interface is utilized for MPHY operation; the remaining 16 bit is utilized for SPHY operation. The content of the Rx_UP_Control_{2..3} registers applies to the SPHY ports. When this 16-bit is configured as a single 16-bit SPHY interface, the content of Rx_UP_Control_3 register is ignored.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																							MS_MODE	PP_MODE	CP_MODE	PARITY_MODE	CELL_SIZE	DR_TIME		

Bits	Field	Description	RW	Reset
[31:7]		reserved	RO	0
[6]	MS_Mode	Master/Slave Mode. When a port is configured for SPHY operation, it can function as either a Rx Master, or as a Tx Slave. 0 = master mode; the port will function as a Rx master 1 = slave mode; the port will function as a Tx slave; This bit is Reserved in Rev A.	RW	0

Bits	Field	Description	RW	Reset
[5]	PP_Mode	<p>When POS-PHY mode is configured (CP_Mode is 1): Used to select between POS-PHY Level 2 or POS-PHY Level 3 operation. The two protocols differ in the way the RXFA status signal is used and have different rules for RXENB assertion and deassertion 0: POS-PHY Level 2 mode: protocol FSM must check RXFA[x] signal 1: POS-PHY Level 3 mode: protocol FSM does not check RXFA[x] signal.</p> <p>When UTOPIA mode is configured (CP_Mode is 0): If the link partner is a UTOPIA Level 1 or 2 device, this bit must always be 0. If the link partner is a UTOPIA Level 3 device, this bit allows the selection of “aggressive RXENB” vs. “conservative RXENB”. This bit only applies to SPHY ports, it has no effect in MPHY mode. Aggressive RXENB mode is recommended, unless the media device cannot handle it. 0: Aggressive RXENB: RXENB will remain asserted unless MSF runs out of space in its receive FIFO. 1: Conservative RXENB: RXENB will be deasserted at the end of every cell transfer, unless RXFA has been asserted, indicating that another cell is available. For Rev A, this bit must be programmed to 0 when UTOPIA mode is configured.</p>	RW	0
[4]	CP_Mode	<p>Cell (UTOPIA) or Packet (POS-PHY) mode: 0: cell 1: packet</p>	RW	0
[3:2]	Parity_Mode	<p>Parity mode: 00: no parity—don’t check incoming parity. The corresponding RXPRTY pin must be tied to 0. 01: Reserved 10: single bit odd parity 11: single bit even parity</p>	RW	0
[1]	Cell_Size	<p>Cell size; only applicable for UTOPIA mode 0: 52 byte cells 1: 53 (x8), 54 (x16), or 56 (x32) byte cells</p>	RW	0
[0]	DR_Time	<p>Decode Response time. This is the time between the following event pairs: RXENB -> RXSOF/RXEOF/RXVAL/RXDATA/RXPRTY RXADDR -> RXPFA 0: 1 clock cycle 1: 2 clock cycles</p>	RW	0

Table 5-57. IXP2400 Rx Mode Programming

Operating Mode	MSF_Rx_Control [Rx_En]				Rx_UP_Control_{0,1,2,3} applicable				Rx_UP_Control_{0,1,2,3} Notes
	[3]	[2]	[1]	[0]	3	2	1	0	
U/P, 1x32, SPHY	0	0	0	1	N/A	N/A	N/A	Y	Rx_UP_Control_0 is used to configure port behavior.
U/P, 1x32, MPHY-4, direct status	1	1	1	1	Y	Y	Y	Y	Rx_UP_Control_0, _1, _2, and _3 must be programmed to identical values; DR_Time should be set to 1 (decode response of two cycles); PP_Mode is not used; MSF_Rx_Control[Rx_MPHY_Level2] must be configured for Level 3 operation.
U/P, 1x32, MPHY-4, polled status	1	1	1	1	Y	Y	Y	Y	Rx_UP_Control_0, _1, _2, and _3 must be programmed to identical values; DR_Time should be set to 1 (decode response of two cycles); PP_Mode is not used; MSF_Rx_Control[Rx_MPHY_Level2] must be configured for Level 3 operation.
U/P, 1x32, MPHY-32, polled status	0	0	0	1	N/A	N/A	N/A	Y	Rx_UP_Control_0 is used to configure port behavior; DR_Time should be set to 1 (decode response of two cycles); MSF_Rx_Control[Rx_MPHY_Level2] must be configured for Level 3 operation.
U/P, 2x16, SPHY	0	1	0	1	N/A	Y	N/A	Y	Rx_UP_Control_{0,2} are used to configure port behavior.
U/P, 1x16 MPHY-32 polled status and 1x16 SPHY (Reserved in Rev A)	0	1	0	1	N/A	Y	N/A	Y	Rx_UP_Control_0 is used to configure the behavior of the x16 MPHY port (DR_Time must be 0); Rx_UP_Control_2 is used to configure the behavior of the x16 SPHY port. MSF_Rx_Control[Rx_MPHY_Level2] must be configured for Level 2 operation.
U/P, 4x8, SPHY	1	1	1	1	Y	Y	Y	Y	Rx_UP_Control_{0,1,2,3} is used to configure port behavior.
U/P, 1x16+2x8, SPHY	1	1	0	1	Y	Y	N/A	Y	Rx_UP_Control_{0,2,3} is used to configure port behavior.
U/P, 1x16 MPHY-32 polled status and 2x8 SPHY (Reserved in Rev A)	1	1	0	1	Y	Y	N/A	Y	Rx_UP_Control_0 is used to configure the behavior of the x16 MPHY port; Rx_UP_Control_{2,3} is are used to configure the behavior of the 2x8 SPHY ports; MSF_Rx_Control[Rx_MPHY_Level2] and MSF_Tx_Control[Tx_MPHY_Level2] must be configured for Level 2 operation.
CSIX, 1x32	0	0	0	1	N/A	N/A	N/A	N/A	Rx_UP_Control_x has no effect in CSIX mode.



5.8.21 Tx_UP_Control_{0..3}

There are four of these registers, one per port. The contents of these registers apply only if the chip is configured for UTOPIA/POS mode.

Note that the Parity_Mode field has a reset value that is Reserved. Users must set it to one of the valid values during the initialization sequence. In case the user does not want parity support, the user can leave the TXPRTY pins unconnected.

For MPHY4 mode configurations, all 4 registers, Tx_UP_Control_{0..3}, must be programmed with the same value. For POS-PHY Level 3 or UTOPIA Level 3 MPHY-32 (MPHY-16 in Rev A) mode configurations, the content in Tx_UP_Control_0 register applies to all the ports, and the content of the Tx_UP_Control_{1..3} registers are ignored. The whole 32-bit interface is utilized in these mode configurations. For POS-PHY Level 2 or UTOPIA Level 2 MPHY-32 (only available in Rev B) mode configurations, the content in Tx_UP_Control_0 register applies to all the MPHY ports, and the content of the Tx_UP_Control_1 register is ignored. In these mode configurations, only 16 bit out of the 32-bit interface is utilized for MPHY operation; the remaining 16 bit is utilized for SPHY operation. The content of the Tx_UP_Control_{2..3} registers applies to the SPHY ports. When this 16-bit is configured as a single 16-bit SPHY interface, the content of Tx_UP_Control_3 register is ignored.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																							MS_MODE	Reserved	CP_MODE	PARITY_MODE	CELL_SIZE	DR_TIME			

Bits	Field	Description	RW	Reset
[0]	DR_Time	Decode Response time: 0: 1 clock cycle 1: 2 clock cycles For IXP2400 Rev A: This bit has no effect for transmit and is here only for consistency with UTOPIA_POS_Rx_Control. For IXP2400 Rev B: This bit configures the response time between TXADDR[4:0] and TXPFA in MPHY modes. For the 32-bit MPHY modes, this bit must be set to 1 for consistency with the UTOPIA 3 and POS-PHY L3 specs. For the 16-bit MPHY modes, this bit can be set to 0 or 1. In this case, setting it to 0 is consistent with the UTOPIA 2 and POS-PHY L2 specs, and setting it to 1 facilitates overclocking of the media interface beyond the typical 50MHz for these protocols. For all SPHY modes, this bit has no effect.	RW	0
[1]	Cell_Size	Cell size; only applicable for UTOPIA mode 0: 52 byte cells 1: 53 (x8), 54 (x16), or 56 (x32) byte cells	RW	0

Bits	Field	Description	RW	Reset
[3:2]	Parity_Mode	Parity mode: 0x: reserved 10: single bit odd parity 11: single bit even parity	RW	0
[4]	CP_Mode	Cell (UTOPIA) or Packet (POS-PHY) mode: 0: cell 1: packet	RW	0
[5]	Reserved		RO	0
[6]	MS_Mode	Master/Slave Mode. When a port is configured for SPHY operation, it can function as either a Tx Master, or as a Rx Slave. 0 = master mode; the port will function as a Tx master 1 = slave mode; the port will function as a Rx slave This bit is Reserved in Rev A.	RW	0
[31:7]	Reserved		RO	0

5.8.22 Rx_FIFO_Control_{0,1,2,3}

There are four of these registers, one per port. The contents of these registers support slave mode operation and apply only when IXP2400 is configured for POS-PHY mode.

In UTOPIA mode, the high watermarks are hardwired by the IXP2400. In CSIX mode, the high watermarks are determined by HWM_Control[RBUF_C_HWM] and HWM_Control[RBUF_D_HWM].

In master mode, the default values should be used. They are identical to the values that are hardwired by the IXP2400 in Rev A.

These registers are Reserved in Rev A.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED														Rx_D_FIFO_FWM				RESERVED		Rx_S_FIFO_HWM										

Bits	Field	Description	RW	Reset
[5:0]	Rx_S_FIFO_HWM	<p>Rx Status FIFO High Watermark; applies only to POS-PHY mode; used to indicate a near full condition in the Rx Status FIFO, which contains Receive Status Words for each received mpacket.</p> <p>In Rx master mode, used to control deassertion of RXENB(m). If the number of entries in the Rx Status FIFO is greater than or equal to the value in this field, the protocol logic will deassert RXENB(m) to stop the flow of data from the PHY.</p> <p>In Tx slave mode, used to control deassertion of TXFA(s). If the number of entries in the Rx Status FIFO is greater than or equal to the value in this field, the protocol logic will deassert TXFA(m) to cause the master to stop sending more transmit data.</p> <p>Note: In Rx_FIFO_Control_2 and Rx_FIFO_Control_3 CSRs, only the lower four bits [3:0] are used; [5:4] are not used.</p> <p>Reset value: RX_FIFO_Control_0 and Rx_FIFO_Control_1: 0x3B Rx_FIFO_Control_2 and Rx_FIFO_Control_3: 0xB</p>	RW	0x3B, 0xB (See Description)
[7:6]	Reserved		RO	0
[15:8]	Rx_D_FIFO_HWM	<p>Rx Data FIFO High Watermark; applies only to POS-PHY mode; used to indicate a near full condition in the Rx Status FIFO, which contains Receive Status Words for each received mpacket.</p> <p>In Rx master mode, used to control deassertion of RXENB(m). If the number of entries in the Rx Data FIFO is greater than or equal to the value in this field, the protocol logic will be asked to deassert RXENB(m) to stop the flow of data from the PHY.</p> <p>In Tx slave mode, used to control deassertion of TXFA(s). If the number of entries in the Rx Data FIFO is greater than or equal to the value in this field, the protocol logic will be asked to deassert TXFA(m) to cause the master to stop sending more transmit data.</p> <p>Note: In Rx_FIFO_Control_2 and Rx_FIFO_Control_3 CSRs, only the lower seven bits [6:0] are used; [7] is not used.</p> <p>Reset value: RX_FIFO_Control_0 and Rx_FIFO_Control_1: 0xF9 Rx_FIFO_Control_2 and Rx_FIFO_Control_3: 0x79</p>	RW	0xF9, 0x79 (See Description)
[31:16]	Reserved		RO	0

5.8.23 MSF_Rx_RCOMP_Status

This register is used to read the RCOMP values of the receive IO buffers. This register is not used during normal operation.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0		
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED													RX_NINC	RX_PINC	RESERVED	RX_N_STRENGTH					RESERVED	RX_P_STRENGTH									

Bits	Field	Description	RW	Reset
[31:18]	reserved	read as 0	RO	0
[17]	Rx_NINC	Output of N Comparator	RO	undef
[16]	Rx_PINC	Output of P Comparator	RO	undef
[15]	reserved	read as 0	RO	0
[14:8]	Rx_n_Strength	Receive RComp_n Strength	RO	undef
[7]	reserved	read as 0	RO	0
[6:0]	Rx_p_Strength	Receive RComp_p Strength	RO	undef

5.8.24 MSF_Tx_RCOMP_Status

This register is used to read the RCOMP values of the transmit IO buffers. This register is not used during normal operation.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0		
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED													TX_NINC	TX_PINC	RESERVED	TX_N_STRENGTH					RESERVED	TX_P_STRENGTH									

Bits	Field	Description	RW	Reset
[31:18]	reserved	read as 0	RO	0
[17]	Tx_NINC	Output of N Comparator	RO	undef
[16]	Tx_PINC	Output of P Comparator	RO	undef
[15]	reserved	read as 0	RO	0
[14:8]	Tx_n_Strength	Transmit RComp_n Strength	RO	undef
[7]	reserved	read as 0	RO	0
[6:0]	Tx_p_Strength	Transmit RComp_p Strength	RO	undef

5.8.25 MSF_Rx_RCOMP_Override

This register is used to override the drive settings of the receive IO buffers. This register is not used during normal operation.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED											RX_RCOMP_OVR_EN	RX_RCOMP_SLEW_OVR			RESERVED	RX_RCOMP_N_OVR					RESERVED	RX_RCOMP_P_OVR									

Bits	Field	Description	RW	Reset
[31:21]	reserved	read as 0	RO	0
[20]	Rx_RComp_OVR_En	Receive RComp Override Enable	RW	0
[19:16]	Rx_RComp_Slew_OVR	Receive RComp Slew Override	RW	0
[15]	reserved	read as 0	RO	0
[14:8]	Rx_RComp_n_OVR	Receive RComp_n Override	RW	0
[7]	reserved	read as 0	RO	0
[6:0]	Rx_RComp_p_OVR	Receive RComp_p Override	RW	0

5.8.26 MSF_Tx_RCOMP_Override

This register is used to override the drive settings of the transmit IO buffers. This register is not used during normal operation.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED											TX_RCOMP_OVR_EN	TX_RCOMP_SLEW_OVR				RESERVED	TX_RCOMP_N_OVR				RESERVED	TX_RCOMP_P_OVR									

Bits	Field	Description	RW	Reset
[31:21]	reserved	read as 0	RO	0
[20]	Tx_RComp_OVR_En	Transmit RComp Override Enable	RW	0
[19:16]	Tx_RComp_Slew_OVR	Transmit RComp Slew Override	RW	0
[15]	reserved	read as 0	RO	0
[14:8]	Tx_RComp_n_OVR	Transmit RComp_n Override	RW	0
[7]	reserved	read as 0	RO	0
[6:0]	Tx_RComp_p_OVR	Transmit RComp_p Override	RW	0

5.8.27 FCIFIFO

This register is used by MEs to read CFrames, one CWord at a time, from the FCIFIFO. When valid CWords are read they are removed from the FCIFIFO. If FCIFIFO is empty when it is read, it substitutes an Idle CWord for the read data.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
CWORD																																

Bits	Field	Description	RW	Reset
[31:0]	CWord	Data from head entry of FCIFIFO, or Idle CWord if FCIFIFO is empty.	RO	0xFF FF

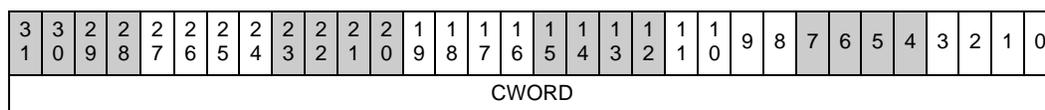
5.8.28 FCEFIFO

This register is used by MEs to write CFrames, one CWord at a time, to the FCEFIFO, when MSF_Rx_Control[Tx_CBus_Mode] is Simplex mode. It is up to the MEs to test for room in FCEFIFO by reading FC_Egress_Status[FCEFIFO_Full].

Software must ensure that all flow-control CFrames (type 0x6) written out this way must have a length that is a multiple of 4 bytes. In addition, software must ensure that the exact number of CWords gets written as specified in the payload length of the corresponding CFrame. Otherwise, undefined behavior may result.

Software must ensure that FCEFIFO does not overflow. Otherwise, the behavior is undefined. For instance, CFrames with corrupted data but valid parity may get transmitted when FCEFIFO overflows.

Software can avoid FCEFIFO overflow by utilizing the FC_Egress_Status[FCEFIFO_Full] bit. Moreover, there is a delay between a CFrame getting written out and the FC_Egress_Status[FCEFIFO_Full] bit reflecting the resulting status. As a result, after sending out some CFrames, software should wait before checking the FC_Egress_Status[FCEFIFO_Full] bit in order to prepare for sending additional CFrames. An upper bound of the amount of time to wait exists but needs to be found out empirically.



Bits	Field	Description	RW	Reset
[31:0]	CWord	Data written to tail entry of FCEFIFO.	WO	0

5.8.29 TBUF_ELEMENT_CONTROL_\$_# (\$= A, B, # = Element No)

This write-only registers are used to set the control information for TBUF elements. The TBUF_ELEMENT_CONTROL is 64-bits and can be address via two registers referred to as “A” and “B”. There is also one set of TBUF_ELEMENT_CONTROL registers per element.

The TBUF_ELEMENT_CONTROL registers are a contiguous block of 128 64-bit (8-byte) registers. When the element size is set to 64 bytes, each TBUF_ELEMENT_CONTROL register is indexed on 8-byte boundaries from the base address specified in Section 4. When the element size is set to either 128 or 256 bytes, the number of TBUF elements is reduced and therefore the number of TBUF_ELEMENT_CONTROL registers required is also reduced. Moreover, the offset for accessing TBUF entries scale by 2x or 4x, depending on whether the TBUF element size is set to 128-byte or 256-byte, respectively. Nevertheless, since the size of TBUF_ELEMENT_CONTROL registers is fixed, the offset or element number for accessing TBUF_ELEMENT_CONTROL registers does not scale up along with the TBUF element size on the IXP2400 network processor. Note that this is different from IXP2800; on IXP2800, the offset for accessing TBUF_ELEMENT_CONTROL registers scales along with the TBUF element size.

Table 5-60. CSIX Transmit Control Word Format

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
Payload Length										Prepend Offset		Prepend Length				Payload Offset			RESERVED	Skip	RESERVED	CR	P	reserved				Type			
6	6	6	6	5	5	5	5	5	5	5	5	5	4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	
3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2
Extension Header																															

Field	Definition
Payload Length	Indicates the number of bytes in the payload, from 1 to 256 bytes, in the element. The value of 0x00 means 256 bytes. The sum of Prepend Length and Payload Length will be sent, and also put into the CSIX base header Length field.
Prepend Offset	Indicates the first valid byte of the prepend, from 0 to 7 bytes.
Prepend Length	Indicates the number of bytes of the prepend from 0 to 31 bytes.
Payload Offset	Indicates the first valid byte of the payload, with respect to the last valid quadword of the prepend.
Skip	Allows software to allocate a TBUF element and then not transmit any data from it. 0: transmit data according to other fields of the Control Word 1: free the element without transmitting any data
CR	CR (CSIX Reserved) bit to put into the CSIX Base Header.
P	P (Private) bit to put into the CSIX Base Header.
Type	Type Field to put into the CSIX Base Header. Idle type is <i>not</i> legal here.
Extension Header	The Extension Header to be sent with the CFrame. For flow control CFrames this field is not used by the hardware because flow control CFrames do not have an extension header.

5.9 PCI

The PCI CSRs are divided into two groups: PCI Configuration Space registers and PCI Control and Status Registers.

5.9.1 PCI Configuration Space

Table 5-61 shows the offset addresses of the PCI Configuration Register set as defined in PCI 2.2. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can be accessed by the Intel XScale® core, PCI and MEs.

Table 5-61. PCI Configuration Register Map

Abbreviation	Address[7:0]	Name	Description	Section
PCI_VEN_DEV_ID	0x00	PCI Device and Vendor Register	Provides Device ID (0x9001 for IXP2400, 0x9004 for IXP2800) and Vendor ID (0x8086)	Section 5.9.1.1
PCI_CMD_STAT	0x04	PCI Command and Status Register	Provides device Status and Command	Section 5.9.1.2
PCI_REV_CLASS	0x08	PCI Class Code Register	Provides chip's Class Code (0x0B4001) and Revision ID (0x0)	Section 5.9.1.3
PCI_CACHE_LAT_HDR_BIST	0x0C	PCI Miscellaneous	Defines BIST, Header Type, Latency Timer, and Cache Line Size fields	Section 5.9.1.4
PCI_CSR_BAR	0x10	PCI Base Address Register for CSRs	Maps CSR address range from PCI	Section 5.9.1.5
PCI_SRAM_BAR	0x14	PCI Base Address Register for SRAM	Maps SRAM address range	Section 5.9.1.6
PCI_DRAM_BAR	0x18	PCI Base Address Register for DRAM	Maps DRAM address range	Section 5.9.1.7
Reserved	0x1C-28			
PCI_SUBSYS	0x2C	PCI Subsystem ID	Provides PCI Subsystem ID and Vendor ID	Section 5.9.1.8
Reserved	0x30-38			
PCI_INT_LAT	0x3C	PCI Interrupt Latency	Defines MAX_LAT, MIN_GNT, INT_PIN, and INT_LINE	Section 5.9.1.9
Reserved	0x40-5C			
PCI_RCOMP_OVERRIDE	0x60	PCI RCOMP Override	Used to override the drive settings of the IO buffers	Section 5.9.1.10
PCI_RCOMP_STATUS	0x64	PCI RCOMP STATUS	Used to the status of the drive settings of the IO buffers	Section 5.9.1.11 Section 5.9.1.12
Reserved	0x68-74			
PCI_IXP_PARAM	0x78	IXP Parameters Register	Special configuration bits for IXP use	Section 5.9.1.13
Reserved	0x7C-FF			

5.9.1.1 PCI_VEN_DEV_ID

This is the Vendor and Device ID register specified in the *PCI Local Bus Specification, Revision 2.2*.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
DEV_ID															VEND_ID																

Bits	Field	Description	RW	Reset
[31:16]	DEV_ID	Device ID. Identifies the IXP2800 / IXP2400 as the device. 0x9004 for IXP2800. 0x9001 for IXP2400.	RO	0x9001 0x9004
[15:0]	VEND_ID	Vendor ID. Identifies Intel as the vendor of this device. Internally hardwired to be 0x8086.	RO	0x8086

5.9.1.2 PCI_CMD_STAT

This is the Command and Status register specified in the *PCI Local Bus Specification, Revision 2.2*.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
PERR_2	SIG_SERR	RX_MA	RX_TA	SIG_TA	DEVSEL[1:0]	PERR	FAST_BACK_T	UDF	66MHZ	RESERVED											FAST_BACK_I	SERR_EN	STEP_EN	PERR_RESP	VGA_EN	WR_INV_EN	SPEC_CYC	BUS_MASTER	MEM_SPACE	IO_SPACE	

Bits	Field	Description	RW	Reset
[31]	PERR_2	Sets if this device detected a parity error in read cycle and even if [6] is cleared.	RW 1C	0
[30]	SIG_SERR	Indicates that this device signalled SERR	RW 1C	0
[29]	RX_MA	Indicates that the external device terminated a transaction with master-abort (time-out) as a master (except for Special Cycle)	RW 1C	0
[28]	RX_TA	Indicates that this device received target-abort as a master.	RW 1C	0
[27]	SIG_TA	Indicates that this device signalled target-abort as a target.	RW 1C	0
[26:25]	DEVSEL[1:0]	Indicates DEVSEL speed for this device. (medium)	RO	01
[24]	PERR	Sets if PERR_RESP is set and as a master, this device either asserted PCI_PERR or saw PCI_PERR asserted for one of its data phases.	RW 1C	0
[23]	FAST_BACK_T	Indicates target is capable of accepting fast back-to-back.	RO	1

Bits	Field	Description	RW	Reset
[22]	UDF	Indicates User Definable Features. Not supported.	RO	0
[21]	66MHZ	Indicates 66 Mhz capability. 66 MHz capable.	RO	1
[20:10]	RESERVED	Reserved	RO	0
[9]	FAST_BACK_I	Enables fast back-to-back transactions to different targets. Note that the integrated PCI controller does not generate fast back-to-back transactions on the PCI bus regardless of the setting of this bit.	RW	0
[8]	SERR_EN	Enables assertion of PCI_SERR. This bit and PERR_RESP must be set to detect and report address parity errors.	RW	0
[7]	STEP_EN	Enables bizarre AD stepping. Not supported.	RO	0
[6]	PERR_RESP	When set, enables PCI parity checking and monitoring of PCI_PERR_L.	RW	0
[5]	VGA_EN	Enables VGA palette snooping. Not supported.	RO	0
[4]	WR_INV_EN	Enables use of Memory Write and Invalidate cycles. Not supported.	RO	0
[3]	SPEC_CYC	Enables response to PCI Special Cycles. Not supported.	RO	0
[2]	BUS_MASTER	Enables PE to act as a PCI master. Must be set to enable DMA.	RW	0
[1]	MEM_SPACE	Enables Mem Space response as a target.	RW	0
[0]	IO_SPACE	Enables IO Space response as a target. Not supported.	RO	0

5.9.1.3 PCI_REV_CLASS

This is the Revision and Class ID register specified in the *PCI Local Bus Specification, Revision 2.2*.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
CLASS								SUB_CLASS								INTERFACE				REV											

Bits	Field	Description	RW	Reset
[31:24]	CLASS	Processor (0x0B)	RO	0x0B
[23:16]	SUB_CLASS	Co-Processor (0x40)	RO	0x40
[15:8]	INTERFACE	Programming Interface (0x01)	RO	0x01
[7:0]	REV	Chip Revision (starts at 0x00, changes with spins if any); Attention to Layout for easy metal patch. For IXP2400 and IXP2800 Rev A, the value is 0x00. For IXP2400 and IXP2800 Rev B, the value is 0x01.	RO	

5.9.1.4 PCI_CACHE_LAT_HDR_BIST

This is the Cache Line Size, Latency Timer, Header Type, and BIST register specified in the *PCI Local Bus Specification, Revision 2.2*.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
BSUP		BSTRT		RESERVED		BCMPT				HDR_TYPE				LAT_TMR_VAL			LAT_TMR_FIX		CACHE_LINE											

Bits	Field	Description	RW	Reset
[31]	BSUP	BIST Support Device. If PROM_BOOT strap = 1: BIST Support If PROM_BOOT strap = 0: BIST is not supported	RW RO	0
[30]	BSTRT	BIST Start Control Bit 0—IXP processor should reset this bit after BIST completes 1—A BIST interrupt is generated and the IXP2800/IXP2400 should reset this bit. If this bit is not reset 2 second after the PCI device sets this bit, the IXP2800/IXP2400 has failed the test.	RW	0
[29:28]	RESERVED	Reserved. Read as 0x0	RO	0
[27:24]	BCMPT	BIST Complete Status 00: IXP processor passed the BIST test. 01 through 0F: IXP processor failed the BIST test. Definable error code.	RW	0
[23:16]	HDR_TYPE	Header format code.	RO	0
[15:11]	LAT_TMR_VAL	Latency timer value. The upper 5-bits of the latency timer as specified by PCI	RW	0
[10:8]	LAT_TMR_FIX	These bits are the low 3 bits of latency timer as specified by PCI. They are hardwired to b000.	RO	0
[7:0]	CACHE_LINE	Cache line size in unit of 32-bit Dwords: Accepts only powers of 2 less than 8 (32 byte). An unsupported value or zero will default to a cacheLine of a single data phase.	RW	0

5.9.1.5 PCI_CSR_BAR

This register is used to specify the base address of the PCI accessible CSRs when using a memory access. The window size is fixed at 1M bytes.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
BASE ADDR												SIZE										TYPE								

Bits	Field	Description	RW	Reset
[31:20]	BASE_ADDR	PCI base address for the CSR. Bits[31:20] determine where the 1M byte window resides in the 4 GB (32-bit) address space.	RW	0
[19:4]	SIZE	Reads as 0x0, writes are ignored.	RO	0
[3:0]	TYPE	Not prefetchable; 32-bit address space; MEM space	RO	0

5.9.1.6 PCI_SRAM_BAR

This register is used to specify the base address of the PCI accessible SRAM when using a memory access. The supported window sizes are 0 (not accessible via PCI), 128Kbytes to 256Mbytes. If the IXP2800 is booted up from the PROM, the SRAM_BASE_ADDR_MASK register determines the window size. Else, if the IXP2800 / IXP2400 is booted up from PCI, the PCI_SWIN strap pins determine the window size. The window size always assumes that there are 4 SRAM channels.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
BASE_ADDR		PROG_ADDR										FIX_ADDR										PREF	TYPE							

Bits	Field	Description	RW	Reset
[31:28]	BASE_ADDR	PCI base address for SRAM	RW	0
[27:18]	PROG_ADDR	Window size determined by BOOT_PROM strap pin 1: SRAM_BASE_ADDR_MASK register programmable from 256 KByte to 256 MBytes and 0 Byte (disabled) 0: Selected by the PCI_SWIN strap pins of 32/64/128/256 MByte. The MASK field in "PCI_SRAM_BAR_MASK" CSR determines if these bits are RW or RO.	*RO /RW	0
[17:4]	FIX_ADDR	Read as 0x0	RO	0
[3]	PREF	Prefetchable space determined by CFG_PROM_BOOT strap pin option. IF CFG_PROM_BOOT strap pin is 1: Program the SRAM_BAR_ADD_MASK register Bit[30] to set or reset the pre-fetch (default is prefetch) 0: Pre-fetch If the DIS bit [31] in the PCI_SRAM_BAR_MASK register is set, this bit will be read as 0	RO	0x1
[2:0]	TYPE	Prefetchable; locatable anywhere in 32-bit address space; MEM space	RO	0x8

5.9.1.7 PCI_DRAM_BAR

This register is used to specify the base address of the PCI accessible DRAM when using a memory access. The supported window sizes are 0 (not accessible via PCI), 1 Mbytes to 1 GBytes. If the IXP2800 is booted up from the PROM, the DRAM_BASE_ADDR_MASK register determines the window size. Else, if the IXP2800 / IXP2400 is booted up from PCI, the PCI_DWIN strap pins determine the window size.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
BASE_ADDR		PROG_ADDR										FIX_ADDR										PREF	TYPE							

Bits	Field	Description	RW	Reset
[31:30]	BASE_ADDR	PCI base address for DRAM	RW	0
[29:20]	PROG_ADDR	Window size determined by CFG_PROM_BOOT strap pin if CFG_PROM_BOOT strap pin is 1: DRAM_BASE_ADDR_MASK register programmable from 1MByte to 1GBytes and 0 Byte (disabled) 0: Selected by PCI_DWIN Strap Pins for 128/256/512/1024 MByte The MASK field in "PCI_SRAM_BAR_MASK" CSR determines if these bits are RW or RO.	*RO /RW	0
[19:4]	FIX_ADDR	Read as 0x0	RO	0
[3]	PREF	Prefetchable space determined by CFG_PROM_BOOT strap pinoption. if CFG_PROM_BOOT strap pin is 1: Program the DRAM_BAR_ADD_MASK register Bit[30] to set or reset the pre-fetch(default is prefetch) 0: Pre-fetch If the DIS bit [31] in the PCI_DRAM_BAR_MASK register is set, this bit will be read as 0	RO	1
[2:0]	TYPE	Locatable anywhere in 32-bit address space; MEM space	RO	0

5.9.1.8 PCI_SUBSYS

This is the Subsystem Vendor ID and Subsystem ID register specified in the *PCI Local Bus Specification Revision 2.2*.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
SID															SVID														

Bits	Field	Description	Intel XScale® Core/ME	PCI	Reset
[31:16]	SID	Subsystem ID. Set by the Intel XScale® core or Microengine software.	RW	RO	0
[15:0]	SVID	Subsystem Vendor ID. Set by the Intel XScale® core or Microengine software	RW	RO	0

5.9.1.9 PCI_INT_LAT

This is the Interrupt Pin, MIN_GNT, and MAX_LAT register specified in the *PCI Local Bus Specification Revision 2.2*.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0				
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
MAX_LAT								MIN_GNT								INT_PIN								INT_LINE							

Bits	Field	Description	RW	Reset
[31:24]	MAX_LAT	Indicates how often this device needs to get to the bus in units of 0.25us. the time between bus requests. This value is used to determine Latency Timer Value. A value of 0 indicates no major requirements for the settings of Latency Timer Value.	RW	0x4
[23:16]	MIN_GNT	Indicates the time needed for a burst, in units of 0.25 uS. This value is used to determine Latency Timer Value. A value of 0 indicates no major requirements for the settings of Latency Timer Value.	RW	0x1
[15:8]	INT_PIN	Indicates which interrupt pin is used. We will connect on INTA#.	RO	0x1
[7:0]	INT_LINE	System interrupt information.	RW	0x0

5.9.1.10 PCI_RCOMP_OVERRIDE

IXP Parameters Register. This register contains configuration bit specific to IXP2400/IXP2800 PCI RCOMP override enable and values.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED											RCOMP_OR_EN	RCOMP_SLEW_OR			RESERVED	N_RCOMP_OR					RESERVED	P_RCOMP_OR						

Bits	Field	Description	RW	Reset
[31:21]	RESERVED	reserved	RO	0x0
[20]	RCOMP_OR_EN	rcomp override enable	RW	0x0
[19:16]	RCOMP_SLEW_OR	rcomp_slew_override 19:18 Enables the N slew over-ride 17:16 Enables the P slew over-ride	RW	0x5
[15]	RESERVED	reserved	RO	0
[14:8]	N_RCOMP_OR	rcomp_n_override. Valid values for the IXP2400 are 0x00 to 0x7F. Valid values for the IXP2800 are 0x00 to 0x0A and values greater than 0x0A will result in the strength being set to 0x05 (1/2 max.).	RW	0x32
[7]	RESERVED	reserved	RO	0
[6:0]	P_RCOMP_OR	rcomp_p_override. Valid values same as N_RCOMP_OR	RW	0x39

5.9.1.11 PCI_RCOMP_STATUS (IXP2400 Rev A and IXP2800)

IXP Parameters Register. This register contains configuration bit specific to IXP2400/IXP2800 PCI RCOMP status values only. Moreover, this definition applies to IXP2400 A-stepping and IXP2800 only.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED											NINC	RESERVED	N_STREN					RESERVED	P_STREN									

Bits	Field	Description	RW	Reset
[31:17]	RESERVED	reserved	RO	0
[16]	NINC	For IXP2400 Rev A and IXP2800 only Indicates the digital output of the comparator in the rcomp buffer. This is from the rcomp buffer called "incb"	RO	undef
[15]	RESERVED	reserved	RO	0
[14:8]	N_STREN	n_Strength Valid values for the IXP2400 are 0x00 to 0x7F. Valid values for the IXP2800 are bits 0x00 to 0x0A.	RO	undef
[7]	RESERVED	reserved	RO	0
[6:0]	P_STREN	p_Strength. Valid values same as N_STREN	RO	undef

5.9.1.12 PCI_RCOMP_STATUS (IXP2400 Rev B)

IXP Parameters Register. This register contains configuration bit specific to IXP2400/IXP2800 PCI RCOMP status values only. Moreover, this definition applies to IXP2400 Rev B only.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED											NINC	PINC	N_SL_EN	P_SL_EN	RESERVED	N_STREN						RESERVED	P_STREN								

Bits	Field	Description	RW	Reset
[31:22]	RESERVED	reserved	RO	0
[21]	NINC	For IXP2400 Rev B only. Indicates the digital output of the comparator in the rcomp buffer. This is from the rcomp buffer called "nincb"	RO	undef
[20]	PINC	For IXP2400 Rev B only. Indicates the digital output of the comparator in the rcomp buffer. This is from the rcomp buffer called "pdcr40"	RO	undef
[19:18]	N_SL_EN	The N slew override	RO	0x1
[17:16]	P_SL_EN	The P slew override	RO	0x1
[16]	NINC	Indicates the digital output of the comparator in the rcomp buffer. This is from the rcomp buffer called "incb"	RO	undef
[15]	RESERVED	reserved	RO	0
[14:8]	N_STREN	n_Strength Valid values for the IXP2400 are 0x00 to 0x7F. Valid values for the IXP2800 are bits 0x00 to 0x0A.	RO	undef
[7]	RESERVED	reserved	RO	0
[6:0]	P_STREN	p_Strength. Valid values same as N_STREN	RO	undef

5.9.1.13 PCI_IXP_PARAM

This register contains configuration bit specific to the IXP2400/IXP2800. The D64 bit allows the IXP2400/IXP2800 to use the PCI 64 bit data path even the PCI system is 32 bit. This allows private connections on the PCI 64 bit extension bus between multiple IXP2400/IXP2800s.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																						TWLEN	RESERVED						DPATH	D64	



Bits	Field	Description	RW	Reset
[31:9]	RESERVED	Reserved	RO	0
[8]	TWLEN	<p>Target write long burst enable</p> <p>1: PCI write burst transactions do not get disconnected at 64-byte address boundaries, unless internal FIFO or buffer is busy.</p> <p>0: PCI write transactions get disconnected at 64-byte address boundaries.</p> <p>This bit/feature is added to IXP2xxx rev B. For rev A, this bit is reserved and the behavior is consistent with setting this bit to 0.</p>	RW	0
[7:2]	RESERVED	Reserved	RO	0
[1]	DPATH	<p>This shows the current PCI Unit 64 bit mode</p> <p>If D64 (bit0 of this register) is:</p> <p>1: The current mode is inversion of the system capability detected</p> <p>0: This bit is the system capability detected</p>	RO	dep
[0]	D64	<p>'1' = attempts D64 transactions if the address is aligned as an initiator. Overrides the 'D64-capable' signal sent by the system at reset time. Used to do 64-bit peer-to-peer communication even if the host is only D32.</p> <p><i>This bit does not show the system capability. If the host is 64-bit capable then setting this bit has no effect. If the host is only 32-bit capable but the backplane allows for 64-bit peer-to-peer activity, setting this bit enables such transactions. There is no penalty for attempting 64-bit transactions in 32-bit-only systems.</i></p> <p>RW 0</p>	RW	0

5.9.2 PCI Controller CSRs

Table 5-62 shows the offset addresses of the PCI CSRs that are specific to the PCI controller (these are not defined by the PCI specification). Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can be accessed the Intel XScale® core, PCI and MEs.

Table 5-62. PCI MEM Space CSR Register Map

Abbreviation	Address [7:0]	Name	Description	Section
PCI_OUT_INT_STATUS	0x30	PCI Outbound Interrupt Status	Provides informations on outstanding interrupts to the PCI Bus	Section 5.9.2.1
PCI_OUT_INT_MASK	0x34	PCI Outbound Interrupt Mask	Provides interrupt masking on interrupts to the PCI Bus	Section 5.9.2.2
MAILBOX_0	0x50	MAILBOX 0	MAILBOX 0	Section 5.9.2.3
MAILBOX_1	0x54	MAILBOX 1	MAILBOX 1	
MAILBOX_2	0x58	MAILBOX 2	MAILBOX 2	
MAILBOX_3	0x5C	MAILBOX 3	MAILBOX 3	
XSCALE_DOORBELL	0x60	XScale Doorbell	Intel XScale® core DOORBELL	Section 5.9.2.4
XSCALE_DOORBELL_SETUP	0x64	XScale Doorbell Setup	Intel XScale® core DOORBELL SETUP	Section 5.9.2.5
PCI_DOORBELL	0x70	PCI Doorbell	PCI DOORBELL	Section 5.9.2.6
PCI_DOORBELL_SETUP	0x74	PCI Doorbell Setup	PCI DOORBELL SETUP	Section 5.9.2.7
CHAN_1_BYTE_COUNT	0x80	Channel 1 DMA Byte Transfer Count	DMA Byte Transfer Count	Section 5.9.2.8
CHAN_1_PCI_ADDR	0x84	Channel 1 DMA PCI Address	DMA PCI Address Register	Section 5.9.2.9
CHAN_1_DRAM_ADDR	0x88	Channel 1 DMA Memory Address	DMA Memory Address	Section 5.9.2.10
CHAN_1_DESC_PTR	0x8c	Channel 1 DMA Descriptor Pointer	DMA Descriptor Pointer	Section 5.9.2.11
CHAN_1_CONTROL	0x90	Channel 1 DMA Control	DMA Control Register for channel owner	Section 5.9.2.12
CHAN_1_ME_PARAM	0x94	Channel 1 Microengine Parameter	DMA Microengine Auto-Push Parameters	Section 5.9.2.13
DMA_INF_MODE	0xE0	DMA Information Mode	Channel Ownership	Section 5.9.2.14
CHAN_2_BYTE_COUNT	0xA0	Channel 2 DMA Byte Transfer Count	DMA Byte Transfer Count	Section 5.9.2.8
CHAN_2_PCI_ADDR	0xA4	Channel 2 DMA PCI Address	DMA PCI Address Register	Section 5.9.2.9
CHAN_2_DRAM_ADDR	0xA8	Channel 2 DMA Memory Address	DMA Memory Address	Section 5.9.2.10
CHAN_2_DESC_PTR	0xAC	Channel 2 DMA Descriptor Pointer	DMA Descriptor Pointer	Section 5.9.2.11
CHAN_2_CONTROL	0xB0	Channel 2 DMA Control	DMA Control Register for channel owner	Section 5.9.2.12

Table 5-62. PCI MEM Space CSR Register Map

Abbreviation	Address [7:0]	Name	Description	Section
CHAN_2_ME_PARAM	0xB4	Channel 2 Microengine Parameter	DMA Microengine Auto-Push Parameters	Section 5.9.2.13
CHAN_3_BYTE_COUNT	0xC0	Channel 3 DMA Byte Transfer Count	DMA Byte Transfer Count	Section 5.9.2.8
CHAN_3_PCI_ADDR	0xC4	Channel 3 DMA PCI Address Register	DMA PCI Address Register	Section 5.9.2.9
CHAN_3_DRAM_ADDR	0xC8	Channel 3 DMA Memory Address	DMA Memory Address	Section 5.9.2.10
CHAN_3_DESC_PTR	0xCC	Channel 3 DMA Descriptor Pointer	DMA Descriptor Pointer	Section 5.9.2.11
CHAN_3_CONTROL	0xD0	Channel 3 DMA Control	DMA Control Register for channel owner	Section 5.9.2.12
CHAN_3_ME_PARAM	0xD4	Channel 3 Microengine Parameter	DMA Microengine Auto-Push Parameters	Section 5.9.2.13
PCI_SRAM_BAR_MASK	0xFC	SRAM Address Mask	Allows BAR window sizing by Intel XScale® core on SRAM	Section 5.9.2.15
PCI_DRAM_BAR_MASK	0x100	DRAM Address Mask	Allows BAR window sizing by Intel XScale® core on DRAM	Section 5.9.2.16
PCI_CONTROL	0x13C	PCI Block CSR	Provides the control and status information in the PCI Blocks	
PCI_ADR_EXT	0x140	PCI Address Extension	Provides the upper address bits for CSR bus direct access to PCI Bus.	Section 5.9.2.18
ME_PUSH_STATUS	0x148	MicroEngine Auto-Push Status	Displaying the pending MicroEngine Auto-Push in progress.	Section 5.9.2.23
ME_PUSH_ENABLE	0x14C	MicroEngine Auto-Push Enable	Masking the Auto-Push to MicroEngine	Section 5.9.2.24
XSCALE_ERR_STATUS	0x150	XScale Error Status Register	Displaying the pending Intel XScale® core Error interrupts.	
XSCALE_ERR_ENABLE	0x154	XScale Error Enable	Masking the Error interrupt to Intel XScale® core	
XSCALE_INT_STATUS	0x158	XScale Interrupt Status Register	Displaying the pending Intel XScale® core interrupts.	Section 5.9.2.21
XSCALE_INT_ENABLE	0x15C	XScale Interrupt Enable	Masking the interrupt to Intel XScale® core	Section 5.9.2.22

5.9.2.1 PCI_OUT_INT_STATUS

This register indicates the reason(s) why the IXP2400/IXP2800 is asserting PCI_INTA_L.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																				DMAD1	DMAD2	DMAD3	RESERVED				WDI	PDI	XSI	PIS	

Bits	Field	Description	RW	Reset
[31:11]	RESERVED	Reserved. Read as 0x0	RO	0
[10]	DMAD1	Channel 1 DMA Done Interrupt: Reads as 1 to indicate that the DMA is done to the host. This is set at the completion of all the descriptors in the link list.	RO	0
[9]	DMAD2	Channel 2 DMA Done Interrupt: Reads as 1 to indicate that the DMA is done to the host. This is set at the completion of all the descriptors in the link list. This bit is reserved for the IXP2800.	RO	0
[8]	DMAD3	Channel 3 DMA Done Interrupt: Reads as 1 to indicate that the DMA is done to the host. This is set at the completion of all the descriptors in the link list.	RO	0
[7:4]	RESERVED	Reserved	RO	0
[3]	WDI	XPI Watchdog timer expired Interrupt. Reads as 1 to indicate that the Watchdog timer expired interrupt from Reset unit when RESET_0[24] (see Section 5.6.4.4) is set to 0 (disable Watchdog timer reset). if IXP RESET_0 Register[24] is set to 1(enable Watchdog timer reset), then Reset unit will not generate XPI Watchdog timer expired Interrupt to PCI.	RO	0
[2]	PDI	PCI Doorbell Interrupt. Reads as 1 to indicate that for at least one bit position of pci doorbell register have been set.	RO	0
[1]	XSI	Intel XScale® core Interrupt. Reads as 1 to indicate a software interrupt from the Intel XScale® core.	RO	0
[0]	PIS	PCI Interrupt Status. Interrupts are re-directed to the PCI Bus when the Intel XScale® core is not present.	RO	0

5.9.2.2 PCI_OUT_INT_MASK

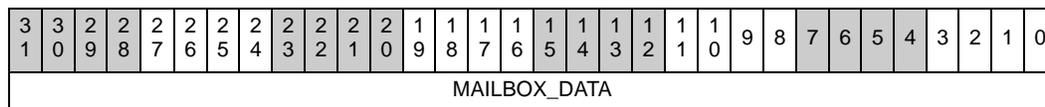
This register allows the host processor to prevent the IXP2400/IXP2800 from asserting the PCI_INTA_L pin. Access from the Intel XScale® processor is not recommended.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																				DMAD_1M	DMAD_2M	DMAD_3M	RESERVED				WDIM	PDIM	XSIM	PISM		

Bits	Field	Description	RW	Reset
[31:11]	RESERVED	Reserved. Read as 0x0	RO	0
[10]	DMAD_1M	Channel 1 DMA Done Interrupt Mask: 0:Enable 1:Disable for the corresponding bits in DMAD	RW	1
[9]	DMAD_2M	Channel 2 DMA Done Interrupt Mask: 0:Enable 1:Disable for the corresponding bits in DMAD This bit is reserved for the IXP2800.	RW	1
[8]	DMAD_3M	Channel 3 DMA Done Interrupt Mask: 0:Enable 1:Disable for the corresponding bits in DMAD	RW	1
[7:4]	RESERVED	Reserved	RO	0
[3]	WDIM	XPI Watchdog timer expired Interrupt Mask. 0:Enable XPI Watchdog timer expired Interrupt 1:Disable XPI Watchdog timer expired Interrupt	RW	1
[2]	PDIM	PCI Doorbell Interrupt Mask 0:Enable doorbell PCI interrupts 1:Disable doorbell PCI interrupts	RW	1
[1]	XSIM	Intel XScale® core Interrupt Mask 0: Enable the interrupt 1: Disable the interrupt	RW	1
[0]	PISM	PCI Interrupt Status Mask 0 Enable re-directed the Intel XScale® core interrupts to the PCI Bus when the Intel XScale® core is not present.	RW	1

5.9.2.3 MAILBOX_#

The # symbol in the name indicates one of 4 mailbox registers (0,1,2, or 3). These registers can be read and written, with byte resolution, form both the Intel XScale® processor, the Microengines and the PCI to exchange messages.



Bits	Field	Description	RW	Reset
[31:0]	MAILBOX_DATA	Mailbox Data: Passes messages between the Intel XScale® core/ME and the host processor. Usage is application dependent. The messages are not used internally by the IXP2800 any way	RW	0

5.9.2.4 XSCALE_DOORBELL

The PCI device writes this register to generate a Doorbell interrupt. Each bit in this register is a write 1 to set from the Intel XScale® core/ME and write 1 to clear from the PCI Bus.

Internally, the logic is such that a value of 0 in a DOORBELL register bit interrupts the Intel XScale® core. The PCI device writes 1 to clear a bit and interrupt the Intel XScale® core and the Intel XScale® core/ME writes 1 to set a bit to clear the interrupt.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
XSCALE_INT_F_HOST																														

Bits	Field	Description	PCI	Intel XScale® Core/ME	Reset
[31:0]	XSCALE_INT_F_HOST	Software Interrupt from host to XScale.	RW 1C	RW 1S	0xffff ffff

5.9.2.5 XSCALE_DOORBELL_SETUP

Doorbell setup is an alias of the doorbell register to allow for initialization and test. It is a read/write register. Data is directly tied to the data input of the Doorbell register.

To initialize the doorbells: Write doorbell setup register such that all bit positions are written with 1.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RW_DATA_ADDR																														

Bits	Field	Description	RW	Reset
[31:0]	RW_DATA_ADDR	Read/Write Data Address. Writes to this address place data directly into the doorbell register. Reads to this address read the value in the Intel XScale® core doorbell register	RW	0x0

5.9.2.6 PCI_DOORBELL

The Intel XScale® processor or the ME writes this register to generate a PCI interrupt. Each bit in this register is a write 1 to set from the Intel XScale® processor or the ME and write 1 to clear from the PCI Bus.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
PCI_SW_INT_T_HOST																														

Bits	Field	Description	PCI	Intel XScale® Core/ME	Reset
[31:0]	PCI_SW_INT_T_HOST	Software Interrupt from the Intel XScale® core or ME to a host on the PCI bus	RW 1C	RW 1S	0

5.9.2.7 PCI_DOORBELL_SETUP

Doorbell setup is an alias of the doorbell register to allow for initialization and test. It is a read/write register. Data is directly tied to the data input of the Doorbell register.

To initialize the doorbells: Write doorbell setup register such that all bit positions used for the Intel XScale® processor or ME to PCI notification are written with 0.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RW_DATA_ADDR																														

Bits	Field	Description	RW	Reset
[31:0]	RW_DATA_ADDR	Read/Write Data Address. Writes to this address place data directly into the doorbell register. Reads to this address read the value in the PCI doorbell register	RW	0x0

5.9.2.8 CHAN_#_BYTE_COUNT

The # symbol in the name indicates the DMA channel (1,2, or 3). The IXP2800 Rev A supports channel 1 only and the IXP2800 Rev B supports channels 1 and 3 only. This register specifies the byte counts for DMA channels 1, 2, and 3.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
EOC	CTD	RESERVED										DMA_BYTE_CNT																			

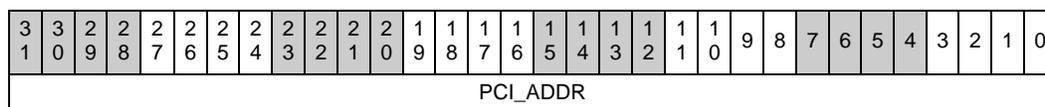
Bits	Field	Description	RW	Reset
[31]	EOC	End Of Chain 0: Indicates that more descriptor list entries follow. 1: Indicates that the current operation is the last in the chain	RW	0

Bits	Field	Description	RW	Reset
[30]	CTD	Control (DMA) Transfer Direction 0: PCI to DRAM 1: DRAM to PCI	RW	0
[29:24]	RESERVED	Reserved. Read as 0x0	R0	0
[23:0]	DMA_BYTE_CNT	DMA Byte Count. Indicates the number of bytes to be transferred. It is updated internally after each read as the DMA operation progresses	RW	0

5.9.2.9 CHAN_#_PCI_ADDR

The # symbol in the name indicates the DMA channel (1,2, or 3). The IXP2800 Rev A supports channel 1 only and the IXP2800 Rev B supports channels 1 and 3 only. These registers contains the PCI address of a DMA transfer. It is the address of the source of data for PCI-to-DRAM transfers, and of the destination of data for DRAM-to-PCI transfers.

It is loaded with the start address of the transfer and is updated internally after each PCI transaction as the transfer proceeds. The initial value does not need to be 32 bit Dword aligned. The PCI byte enables is adjusted according to the two low-order bits of the address. After the first PCI access, the updated value is 32 bit aligned.

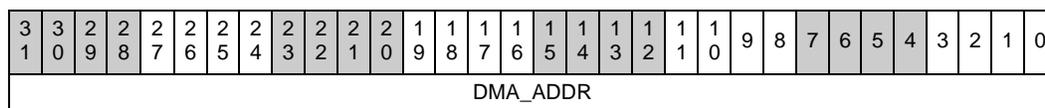


Bits	Field	Description	RW	Reset
[31:0]	PCI_ADDR	PCI Address. Contains the address on the PCI bus for reads or writes. It is updated internally as the DMA operation progresses.	RW	0

5.9.2.10 CHAN_#_DRAM_BAR

The # symbol in the name indicates the DMA channel (1,2, or 3). The IXP2800 Rev A supports channel 1 only and the IXP2800 Rev B supports channels 1 and 3 only. These registers contain the DRAM address of a DMA transfer. It is the address of the source of data for DRAM-to-PCI transfers, and of the destination of data for PCI-to-DRAM transfers.

It is loaded with the start address of the transfer and is updated internally after each DRAM transaction as the transfer proceeds. The initial value does not need to be 32 bit Dword aligned.



Bits	Field	Description	RW	Reset
[31:0]	DMA_ADDR	DRAM Address. Contains the address of the DRAM for reads or writes. It is updated internally as the DMA operation progresses.	RW	0

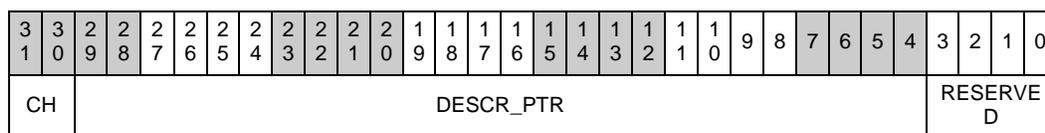
5.9.2.11 CHAN_#_DESC_PTR

The # symbol in the name indicates the DMA channel (1,2, or 3). The IXP2800 Rev A supports channel 1 only and the IXP2800 Rev B supports channels 1 and 3 only. These registers contain the SRAM address of the next DMA descriptor for this channel. If the end-of chain bit in the DMA channel n byte count register is 0, the DMA channel reads the next descriptor from SRAM when the current transfer is done.

The descriptor must be 16 byte aligned, that is, the three low-order bits of the address must be 0. See [Table 5-63](#).

Table 5-63. Descriptor Format:

Offset from Descriptor Pointer	Description
0x0	Byte Count
0x4	PCI Address
0x8	DRAM Address
0xC	Next Descriptor Address



Bits	Field	Description	RW	Reset
[31:30]	CH	SRAM Channel number (0,1,2,3)	RW	0
[29:4]	DESCR_PTR	Descriptor Pointer: Contains the address of the next descriptor in SRAM. This field contains an address expressed in terms of quad DWords (16 bytes); bits 29:0 are viewed as a byte address. This field represents the offset from the base of the SRAM channel specified in the CH field [31:30]. Note that the current address space for each SRAM channel is 64M byte. As such, users should ensure that bits [29:26] are 0.	RW	0
[3:0]	RESERVED	Reserved. Read as 0. Effectively aligns the descriptor address on a 16 byte boundary	RO	0

5.9.2.12 CHAN_#_CONTROL

The # symbol in the name indicates the DMA channel (1,2, or 3). The IXP2800 Rev A supports channel 1 only and the IXP2800 Rev B supports channels 1 and 3 only. These registers contain values that control the DMA channels for the duration of a chain operation. This register must be written at the beginning of the channel operation. It can be read to monitor status of the channel.

Unlinked (or Unterminated) Descriptor is a descriptor with End of Chain bit =0 and the Next Descriptor field = 0. This descriptor is not the last one in the chain, but the Next Descriptor field value of zero is reserved as an indication that the following descriptor has not yet been put into memory. This feature allows software to start a channel with only part of the descriptor chain defined. If the channel reads the Unlinked Descriptor prior to software adding to the chain, it will pause. In addition, the channel will not update the Next Descriptor register with the 0 value; i.e. it will retain the prior value. This allows the channel to fetch the new descriptor after software has linked it into the chain and notified the channel by writing the DA bit below.

The UDR, Paused, and DA don't really need to be visible via a CSR read. They are readable for debug purpose. The channel's action when a transfer completes is based on the state of the Unlinked Descriptor Read, Paused and Descriptor Added bits (see [Table 5-64](#)):

Table 5-64. Operation of Unlinked Descriptor

Current Value			Next Value			Comments
UDR	Paused	DA	UDR	Paused	DA	
0	0	0	0	0	0	No new descriptor was added. Next is already valid
0	0	1	0	0	0	New descriptor was linked in before channel has read the Unlinked Descriptor. Next is already valid. Clear DA.
1	0	x	1	1	x	New descriptor was not yet linked after channel read Unlinked descriptor. Go to Paused state.
1	1	0	1	1	0	Channel is in Paused state. Wait here for DA
1	1	1	0	0	0	Channel rereads the descriptor and continues
0	1	x	x	x	x	This condition cannot happen.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0					
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
CHAN_XFER_DONE_CNT																UDR	PAUSED	DA	RESERVED	DWLEN	DRLLEN	CCD	RESERVED	CIDR	CH_ERR	CXD	RESERVED	CE			

Bits	Field	Description	RW	Reset
[31:16]	CHAN_XFER_DONE_CNT	Channel Transfer Done Descriptor Count. Indicates the number of descriptors whose transfer count have reached 0. The count will be reset after read of this register.	RW1S	0
[15]	UDR	Set by channel when it reads an Unlinked descriptor. Note this bit is set when the descriptor is read, not when the descriptor is completed.	RO	0
[14]	PAUSED	Set by channel when it completes an Unlinked descriptor. This indicates that the channel is waiting for software to add another descriptor.	RO	0
[13]	DA	Software writes a 1 to this bit to notify the channel that it has linked a new descriptor onto an Unlinked descriptor.	RW	0

Bits	Field	Description	RW	Reset
[12:10]	RESERVED	Reserved. Read as 0x0	RO	0
[9]	DWLEN	<p>DMA write long burst enable</p> <p>0: Maximum burst size is 64-byte</p> <p>1: For DMA writes to DRAM from PCI, long burst transactions are requested beginning with the starting 64-byte aligned address and up to the last 64-byte aligned address. Once such long burst transactions start execution, transactions from other DMA channels and from commands that target the PCI address space will not get serviced until the end of the burst read from PCI finishes.</p> <p>This bit/feature is added to the IXP2xxx rev B. For rev A, this bit is reserved, and the behavior is consistent with setting this bit to 0.</p>	RW	0
[8]	DRLEN	<p>DMA read long burst enable</p> <p>0: Maximum burst size is 64-byte</p> <p>1: For DMA transfers from DRAM to PCI space, the PCI unit continues the burst transfer as long as the data for the following 64-byte of address range is already read from DRAM and is available in the buffer of the PCI unit.</p> <p>This bit/feature is added to the IXP2xxx rev B. For rev A, this bit is reserved, and the behavior is consistent with setting this bit to 0.</p>	RW	0
[7]	CCD	Channel chain done. Indicates that a chain has completed either normally or due to an error condition. When this bit is a 1, it can interrupt DMA initiator depending on the ownership of the channel	RW1C	0
[6:5]	RESERVED	Reserved. Read as 0x0	RO	0
[4]	CIDR	<p>Channel initial descriptor in register</p> <p>0: Indicates that the channel must read the first descriptor from SRAM</p> <p>1: Indicates that the DMA channel owner has written the first descriptor to the channel registers. Channel reads of subsequent descriptors, if any, are not affected by this bit.</p>	RW	0
[3]	CH_ERR	<p>Channel error. Indicates that the channel detected either a PCI master abort, target abort, or parity error during a PCI transfer, or bad parity during a DRAM read.</p> <p>1: When set, the channel stops operation regardless of the byte count and/ or end-of-chain bits.</p>	RW1C	0
[2]	CXD	<p>Channel transfer done.</p> <p>1: Indicates that a transfer has completed, that is, the transfer count from one of the descriptors has reached 0.</p>	RW1C	0
[1]	RESERVED	Reserved	RO	0
[0]	CE	<p>Channel enable</p> <p>0: Channel not active</p> <p>When written from 0 to 1: Channel fetches the first descriptor block (unless channel initial descriptor in register bit 4 of this register is a 1), and then performs DMA operations until it does a transfer with the end-of-chain bit equal to 1. This bit is cleared internally when channel chain done is set.</p>	RW	0

5.9.2.13 CHAN_#_ME_PARAM

The # symbol in the name indicates the DMA channel (1,2, or 3). Rev A of the IXP2800 only supports channel 1. Rev B of the IXP2800 supports channels 1 and 3. IXP2400 supports all three channels. This register is to record the information on the Microengine that sets up the DMA channel. This allows the DMA engine to signal a Microengine thread at the completion of a DMA. Specifically, the snapshot of the CHAN_#_CONTROL (# for the same DMA channel) at the DMA completion is copied into a ME Transfer register and a ME signal is set. The ME thread, signal, and transfer register to be used are specified in the ME_CLUS, ME_NO, CTX_NO, REG_NO, and SIG_NO fields of CHAN_#_ME_PARAM.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
CTX	RESERVED													ME_CLUS	RESERVED	ME_NO	CTX_NO	REG_NO	SIG_NO								

Bits	Field	Description	RW	Reset
[31]	CTX_MODE	Number of ME contexts: 0: 8 CTX mode 1: 4 CTX mode	RW	0
[30:17]	RESERVED	Reserved. Read as 0.	RO	0
[16]	ME_CLUS	ME Cluster	RW	0
[15]	RESERVED	Reserved	RW	0
[14:12]	ME_NO	Microengine number that will be signaled. Valid IXP2800 ME numbers are 0 - 7. Valid IXP2400 ME numbers are 0 - 3.	RW	0
[11:9]	CTX_NO	Context Number	RW	0
[8:4]	REG_NO	Register Number	RW	0
[3:0]	SIG_NO	Signal Number	RW	0

5.9.2.14 DMA_INF_MODE

This register specifies whether a PCI device, the MEs, or the Intel XScale® processor owns one of both DMA channels.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																							CH3	MODE			

Bits	Field	Description	RW	Reset
[31:4]	RESERVED	Reserved	RO	0
[3:2]	CH3	DMA channel allocation: 11: Reserved 10: PCI owns DMA channel 3 01: Intel XScale® core owns DMA channel 3 00: Microengine owns DMA channel 3. This field is reserved in IXP2800 Rev A	RW	0x1
[1:0]	MODE	DMA channel allocation for the first two channels For the IXP2400 (supports channels 1, 2, and 3): 11: PCI owns both DMA channels 1 and 2 10: Microengine owns both DMA channels 01: Intel XScale® core owns both DMA channels 00: Microengine owns DMA channel 2 and Intel XScale® core owns DMA channel 1. For the IXP2800 Rev A (supports channel 1 only): 11: PCI own channels 1 10: Microengine own channels 1 01: Intel XScale® core own channels 1 00: Reserved wns DMA channel 1.	RW	0x1

5.9.2.15 PCI_SRAM_BAR_MASK

This register is used to specify the window size for the PCI_SRAM_BAR register.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
DIS		NPR		RESERVED		MASK										RESERVED												

Bits	Field	Description	RW	Reset
[31]	DIS	SRAM Window Disable This bit set plus the MASK bits in this register indicates that the SRAM is not accessible via the PCI	RW	0
[30]	NPR	Non prefetchable This bit set indicates that the SRAM BAR is non prefetchable	R/W	0
[29:28]	RESERVED	Reserved	RO	0
[27:18]	MASK	Address Window Size Mask 1—the corresponding bit in the BAR register is a read-only 0 bit. 0—the corresponding bit in the BAR register is a read/write bit.	RW	0
[17:0]	RESERVED	Reserved	RO	0

The window sizes are determined as shown in [Table 5-65](#):

Table 5-65. How Window Sizes are Determined (PCI_SRAM_BAR)

Window Size	Valid Values for PCI_SRAM_BAR Mask	Bits that are RW in PCI_SRAM_BAR
256 Kbytes	0x0000 0000	31:18
512 Kbytes	0x0004 0000	31:19
1 Mbyte	0x000C 0000	31:20
2 Mbyte	0x001C 0000	31:21
4 Mbyte	0x003C 0000	31:22
8 Mbyte	0x007C 0000	31:23
16 Mbyte	0x00FC 0000	31:24
32 Mbyte	0x01FC 0000	31:25
64 Mbyte	0x03FC 0000	31:26
128 Mbyte	0x07FC 0000	31:27
256 Mbyte	0x0FFC 0000	31:28
0 bytes	0x8FFC 0000	All bits are read as 0

5.9.2.16 PCI_DRAM_BAR_MASK

This register is used to specify the window size for the PCI_DRAM_BAR register.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
DIS	NPR	MASK											RESERVED																

Bits	Field	Description	RW	Reset
[31]	DIS	DRAM Window Disable This bit set plus the MASK bits in this register indicates that the DRAM is not accessible via the PCI	RW	0
[30]	NPR	Non prefetchable This bit set indicates that the DRAM BAR is non prefetchable	R/W	0
[29:20]	MASK	Address Window Size Mask 1—the corresponding bit in the BAR register is a read-only 0 bit. 0—the corresponding bit in the BAR register is a read/write bit.	RW	0
[19:0]	RESERVED	Reserved	RO	0

The window sizes are determined as shown in [Table 5-66](#):

Bits	Field	Description	RW	Reset
[31]	CFG_PCI_BOOT	Boot Host. This bit reflects the value on the CFG_PCI_BOOT pin; 1—IXP will configure the system 0—External host will configure the system If CFG_PCI_BOOT_HOST is set to 1 then CFG_RSTDIR must also be set to 1. (central function)	RO	-
[30]	CFG_PROM_BOOT	PCI Prom Boot. This bit reflects the value on the CFG_PCI_BOOT_HOST pin; 1—IXP will boot from PROM 0—IXP will boot from DRAM initialized by PCI Host.	RO	-
[29]	CFG_PCI_ARB	PCI Internal Arbiter pin status. This bit reflects the value on the CFG_PCI_ARB pin; 1—IXP's internal arbiter is used If CFG_PCI_ARB set to 1 then CFG_RSTDIR must set to 1 (central function).	RO	-
[28]	CFG_RST_DIR	PCI central function pin: 1—IXP is supporting central functions. PCI_RST_L is an output. (PCI_RST_L is output and SYS_RST_L is input). PCI_REQ64 is an output.(drive low during PCI reset) PCI AD[31:0], PCI_BE[3:0], and PCI_PAR drive to Low during PCI reset After PCI reset all these I/O to tristates unless IXP bus parked. PCI AD[63:32], BE[7:4] and PAR64 during PCI reset or after PCI reset all these I/O to tristates. if PCI is 32 bits bus, Board need to support external pull up). 0—External PCI is supporting central functions. PCI_RST_L is an input. (Both PCI_RST_L and SYS_RST_L are input. Tie both reset together). PCI_REQ64 is an input. PCI AD[31:0], PCI_BE[3:0], and PCI_PAR during PCI reset or after PCI reset all these I/O to tristates PCI AD[63:32], BE[7:4] and PAR64 during PCI reset or after PCI reset all these I/O to tristates. (if PCI is 32 bits bus, Board need to support external pull up) This pin is stored at XSC[31] (XSCALE_CONTROL Register) at the trailing edge of reset.	RO	-
[27]	DMA_IDLE	DMA Idle: This bit indicates the DMA Engine is in Idle State	RO	-
[26]	PIN_IN_RST	PCI Bus is in reset; All access to PCI Bus will return bad data on read and data discarded on writes	RO	0
[25]	TABT_DISABLE	Target Abort on PCI Bus disable	RW	0
[24]	XS_INT	Intel XScale® core Interrupt: Software writes to this bit to perform a soft interrupt to PCI	RW	0
[23]	RESERVED	Reserved. Software should not set this bit to 1, because it is reserved.	RW	0

Bits	Field	Description	RW	Reset
[22]	BE_DEO	<p>Big Endian Data Enable Out—When set, data swapping is not performed on the outgoing PCI data during:</p> <ul style="list-style-type: none"> External PCI Master reads from SRAM or DRAM DMA channel writes to PCI target Command Target writes to PCI Mem Space <p>Memory data [31:0] = {byte0,byte1,byte2,byte3}</p> <p>0: PCI data [31:0] = {byte3,byte2,byte1,byte0}</p> <p>1: PCI data [31:0] = {byte0,byte1,byte2,byte3}</p> <p>Supports I/O cycles in Rev B ONLY. See PCI_CONTROL[17]</p>	RW	0
[21]	BE_DEI	<p>Big Endian Data Enable In—When set, data swapping is not performed on the incoming PCI data during:</p> <ul style="list-style-type: none"> External PCI Master writes from SRAM or DRAM DMA channel reads from PCI target Command Target reads from PCI Mem Space <p>Memory data [31:0] = {byte0,byte1,byte2,byte3}</p> <p>0: PCI data [31:0] = {byte3,byte2,byte1,byte0}</p> <p>1: PCI data [31:0] = {byte0,byte1,byte2,byte3}</p> <p>Supports I/O cycles in Rev B ONLY. See PCI_CONTROL[17]</p>	RW	0
[20]	BE_BEO	<p>Big Endian Byte Enable Out—When set, byte enable swapping is performed on the outgoing PCI byte enables during:</p> <ul style="list-style-type: none"> External PCI Master reads from SRAM or DRAM DMA channel writes to PCI target Command Target writes to PCI Mem Space <p>Memory byte enable [3:0] = {be0,be1,be2,be3}</p> <p>0: PCI byte enable [3:0] = {be3,be2,be1,be0}</p> <p>1: PCI byte enable [3:0] = {be0,be1,be2,be3}</p> <p>Supports I/O cycles in Rev B ONLY. See PCI_CONTROL[17]</p>	RW	0
[19]	BE_BEI	<p>Big Endian byte enable Enable In—When set, byte enable swapping is performed on the incoming PCI byte enable during:</p> <ul style="list-style-type: none"> External PCI Master writes from SRAM or DRAM DMA channel reads from PCI target Command Target reads from PCI Mem Space <p>Memory byte enable [3:0] = {be0,be1,be2,be3}</p> <p>0: PCI byte enable [3:0] = {be3,be2,be1,be0}</p> <p>1: PCI byte enable [3:0] = {be0,be1,be2,be3}</p> <p>Supports I/O cycles in Rev B ONLY. See PCI_CONTROL[17]</p>	RW	0
[18]	ATWE	<p>For IXP2xxx Rev B only. Reserved otherwise.</p> <p>Atomic write enable</p> <p>When set, PCI burst target writes to memory that are 64-byte aligned and are less than or equal to 64 bytes long are processed atomically. One implication is that the size of the queue that keeps track of outstanding PCI target write transactions gets reduced to 1.</p> <p>When cleared, PCI burst target writes may not be processed atomically, and the queue that keeps track of PCI target write transactions can track up to 4 outstanding transactions.</p>	RO, RW for IXP2400 Rev B	0

Bits	Field	Description	RW	Reset
[17]	IEE	For IXP2xxx Rev B only. Reserved otherwise. I/O cycles big endian data swap enable When set, the big endian data swapping functionalities of the BE_DEO, BE_DEI, BE_BEO, and BE_BEI fields also apply to PCI I/O cycles. When cleared, the byte ordering of PCI I/O accesses is not affected by the BE_DEO, BE_DEI, BE_BEO, and BE_BEI fields, and is the same as the byte ordering of CSR accesses.	RO, RW for IXP2400 Rev B	0
[16]	DTE	Discard Timer expired. Set when the discard timer counts to 0 and the IXP processor has discarded read data. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register. To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.	RW1C	0
[15]	TRB	TGT_REG_BE: A Target access of Registers detected an address byte enable error Only 32 bit data with all the byte enables set or cleared is allowed for accesses to PCI_CSR_BAR. No data is written on write and garbage data is returned on error. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register. To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.	RW 1C	0
[14]	TWR	TGT_WR_PAR - Target write with bad data. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register. To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.	RW1C	0
[13]	TAE	TGT_ADR_ERR-Detected Address Parity Error; would have generated PCI_SERR#. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register. To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.	RW1C	0
[12]	TDE	TGT_DRAM_ERR-Detected Parity Error on data from the DRAM. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register. To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.	RW1C	0
[11]	TSE	TGT_SRAM_ERR-Detected Parity Error on data from the SRAM. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register. To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.	RW1C	0
[10]	DDE	DMA_DRAM_ERR-Detected Parity Error on data from the DRAM during data transfer from DRAM. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register. To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.	RW1C	0

Bits	Field	Description	RW	Reset
[9]	DSE	<p>DMA_SRAM_ERR-Detected Parity Error on data from the SRAM during descriptor fetch. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register.</p> <p>To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.</p>	RW1C	0
[8]	DCT	<p>DCT is used to indicate that an access to the PCI Bus from the Intel XScale® core or ME has received an error. Because it is a single bit indication, the type of error is indicated by other error bits (RTA, RMA, RSE, DPE, DPED)</p> <p>This bit does not generate an interrupt, but it stops the Intel XScale® core/ME PCI accesses. The bit has to be cleared for the state machine to operate for ME and Intel XScale® core accesses.</p> <p>The use of this bit is to separate the direct access error from DMA errors. When an interrupt is generated due to an error, software can read this bit to tell if it is from the Intel XScale® core/ME generated cycle or not.</p> <p>If any of the following errors were set during a PCI access from the Intel XScale® core or the Microengine, this bit will be set. The errors are RTA, RMA, RSE, DPE, DPED</p> <p>To clear this bit, software must write a 1 to this bit. Writing a 0 to this bit has no effect.</p>	RW1C	0
[7]	DPE	<p>Detected PCI write parity error. When PCI device write to external PCI bus, PCI device will detect the parity error in this write cycle and set this bit. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register.</p> <p>To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.</p>	RW1C	0
[6]	RTA	<p>Received target abort. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register.</p> <p>To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.</p>	RW1C	0
[5]	RMA	<p>Received master abort. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register.</p> <p>To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.</p>	RW1C	0
[4]	DPED	<p>MST_RD_PAR - Data Parity error detected when PCI Unit is initiating a PCI read. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register.</p> <p>To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.</p>	RW1C	0
[3]	RS	<p>Received SERR. Set when PCI SERR is asserted by an external device and CFG_PCI_BOOT (hosting) is asserted. It can cause an interrupt to the Intel XScale® core if enabled in the XSCALE_ERR_ENABLE register.</p> <p>To clear this error, software must write a 1 to this bit. Writing a 0 to this bit has no effect.</p>	RW1C	0

Bits	Field	Description	RW	Reset
[2]	PCII	PCI Interrupt to Intel XScale® core: Software write to this bit to perform a soft interrupt to Intel XScale® core.	RW	0
[1]	AS	Assert SERR. 0: Reset value 0. 1: The PCI Unit asserts PCI_SERR# for one cycle if CFG_PCI_BOOT is de-asserted (not hosting) and command register bit SERR# enable bit [8] is a 1 (PCI_CMD_STAT[SERR])	RW	0
[0]	CA	Clock Active: This indicates the PCI clock is running on the PCI Bus. 0: No PCI Clock 1: PCI Clock is running. The Intel XScale® core/ME can clear initialization bit	RO	0

5.9.2.18 PCI_ADDR_EXT

This register contains the upper PCI address bits during PCI I/O or memory accesses originated by the Intel XScale® processor and MEs.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
PIO ADD														PMSA			Reserved												

Bits	Field	Description	RW	Reset
[31:16]	PIO ADD	PCI IO space upper Address Field. These bits provide PCI address [31:16] during the Intel XScale® core and ME originated PCI IO accesses	RW	0
[15:13]	PMSA	PCI Memory Space Address Field. This bit provides PCI address [31:29] during the Intel XScale® core and ME originated PCI memory accesses.	RW	0
[12:0]	Reserved	Read as 0x0	RO	0

5.9.2.19 XSCALE_ERR_STATUS

A 1 in any of bits [31:19] of this register indicates that the associated error source is active in generating the interrupt to the Intel XScale® core. The interrupt is reported to bit [25], PCI_ERR, of the ERR_RAW_STATUS register. These bits can be cleared by writing to the PCI_CONTROL register and the PCI_CMD_STAT register

Bits [11:0] only apply to IXP2xxx Rev B, which includes the capability to optionally redirect the Intel XScale® core interrupts as PCI_INTA_L. These bits reflect some of the bits of the {IRQ,FIQ}ERR_STATUS Intel XScale® core local CSR, when the corresponding bits in the XSCALE_ERR_ENABLE register are set

Bits	Field	Description	RW	Reset
[26]	RSERR	Received SERR This error is cleared by writing 1 to the following bits: <ul style="list-style-type: none"> • PCI_CONTROL [3] • PCI_CONTROL [8] if the error resulted from a Microengine or an Intel XScale® core direct transaction, 	RO	0
[25]	TRB	TGT_REG_BE - A Slave access of Registers detected a byte enable error This error is cleared by writing 1 to PCI_CONTROL [15]	RO	0
[24]	TWP	TGT_WR_PAR Target write with bad data This error is cleared by writing 1 to PCI_CONTROL [14]	RO	0
[23]	TAE	TGT_ADR_ERR Detected Address Parity Error; would have generated PCI_SERR# if command register rserr enable bit is set This error is cleared by writing 1 to PCI_CONTROL [13]	RO	0
[22]	TDE	TGT_DRAM_ERR - Detected Parity Error on data from the DRAM This error is cleared by writing 1 to PCI_CONTROL [12]	RO	0
[21]	TSE	TGT_SRAM_ERR - Detected Parity Error on data from the SRAM This error is cleared by writing 1 to PCI_CONTROL [11]	RO	0
[20]	DDE	DMA_DRAM_ERR - Detected Parity Error on data from the DRAM This error is cleared by writing 1 to PCI_CONTROL [10]	RO	0
[19]	DSE	DMA_SRAM_ERR- Detected Parity Error on data from the SRAM This error is cleared by writing 1 to PCI_CONTROL [9]	RO	0
[18:12]	Reserved	Read as 0x0	RO	0
[11]	SP_INT	Slow Port interrupt. To clear, write 1 to the Slow Port's fault status register. For IXP2xxx Rev B only. Otherwise Reserved.	RO	0
[10]	MEDIA_ERR	Media error indicator. To clear, write 1 to the error bit in the MSF interrupt status register. For IXP2xxx Rev B only. Otherwise Reserved.	RO	0
[9]	SRAM3_ERR	Indicates a SRAM parity error has occurred in SRAM channel 3. To clear, write 1 to the error bit in the SRAM3 parity status register. For IXP28xx Rev B only. Otherwise Reserved.	RO	0
[8]	SRAM2_ERR	Indicates a SRAM parity error has occurred in SRAM channel 2. To clear, write 1 to the error bit in the SRAM2 parity status register. For IXP28xx Rev B only. Otherwise Reserved.	RO	0
[7]	SRAM1_ERR	Indicates a SRAM parity error has occurred in SRAM channel 1. To clear, write 1 to the error bit in the SRAM1 parity status register. For IXP2xxx Rev B only. Otherwise Reserved.	RO	0

Bits	Field	Description	RW	Reset
[6]	SRAM0_ERR	Indicates a SRAM parity error has occurred in SRAM channel 0. To clear, write 1 to the error bit in the SRAM0 parity status register. For IXP2xxx Rev B only. Otherwise Reserved.	RO	0
[5]	DRAM2_EC_C_MAJ	Uncorrectable ECC error occurred in DRAM channel 2. To clear, write 1 to the error bit in the DRAM2 error status register. For IXP2800 Rev B only. Otherwise Reserved.	RO	0
[4]	DRAM2_EC_C_MIN	Correctable ECC error occurred in DRAM channel 2. To clear, write 1 to the error bit in the DRAM2 error status register. For IXP2800 Rev B only. Otherwise Reserved.	RO	0
[3]	DRAM1_EC_C_MAJ	Uncorrectable ECC error occurred in DRAM channel 1. To clear, write 1 to the error bit in the DRAM1 error status register. For IXP2800 Rev B only. Otherwise Reserved.	RO	0
[2]	DRAM1_EC_C_MIN	Correctable ECC error occurred in DRAM channel 1. To clear, write 1 to the error bit in the DRAM1 error status register. For IXP2800 Rev B only. Otherwise Reserved.	RO	0
[1]	DRAM0_EC_C_MAJ	Uncorrectable ECC error occurred in DRAM channel 0. To clear, write 1 to the error bit in the DRAM0 error status register. For IXP2xxx Rev B only. Otherwise Reserved.	RO	0
[0]	DRAM0_EC_C_MIN	Correctable ECC error occurred in DRAM channel 0. To clear, write 1 to the error bit in the DRAM0 error status register. For IXP2xxx Rev B only. Otherwise Reserved.	RO	0

5.9.2.20 XSCALE_ERR_ENABLE

This register is used to mask the interrupt input sources and define which active sources generate an interrupt request to the Intel XScale® core. A one in a particular bit location will enable that interrupt.

Bits [11:0] only apply to IXP2xxx Rev B, which includes the capability to optionally redirect Intel XScale® core interrupts as PCI_INTA_L. These bits enable the redirection of the Intel XScale® core interrupt that result from the individual error conditions. Note that the PCI Interrupt Mask bit of the PCI_OUT_INT_MASK register must be 0 to enable the redirection of interrupts to PCI.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0					
DPEM	RTAM	RMAM	DPEDM	DTEM	RSERRM	TRBM	TWPM	TAEM	TDEM	TSEM	DDEM	DSEM	Reserved										SP_INTE	MEDIA_ERRE	SRAM3_ERRE	SRAM2_ERRE	SRAM1_ERRE	SRAM0_ERRE	DRAM2_ECC_MAJE	DRAM2_ECC_MINE	DRAM1_ECC_MAJE	DRAM1_ECC_MINE	DRAM0_ECC_MAJE	DRAM0_ECC_MINE



Bits	Field	Description	RW	Reset
[31]	DPEM	Detected PCI parity error	RW	0
[30]	RTAM	Received target abort	RW	0
[29]	RMAM	Received master abort	RW	0
[28]	DPEDM	Data Parity error detected	RW	0
[27]	DTEM	Discard timer expired	RW	0
[26]	RSERRM	Received SERR	RW	0
[25]	TRBM	A Slave access of Registers detected an address byte enable error	RW	0
[24]	TWPM	Target write with bad data	RW	0
[23]	TAEM	Detected Address Parity Error; would have generated PCI_SERR#	RW	0
[22]	TDEM	Detected Parity Error on data from the DRAM	RW	0
[21]	TSEM	Detected Parity Error on data from the SRAM	RW	0
[20]	DDEM	Detected Parity Error on data from the DRAM	RW	0
[19]	DSEM	Detected Parity Error on data from the SRAM	RW	0
[18:12]	Reserved	Read as 0x0	RO	0
[11]	SP_INTE	Slow Port interrupt redirect enable. For IXP2xxx Rev B only. Otherwise Reserved.	RO, RW for IXP2xxx Rev B	0
[10]	MEDIA_ER RE	Media error indicator interrupt redirect enable. For IXP2xxx Rev B only. Otherwise Reserved.	RO, RW for IXP2xxx Rev B	0
[9]	SRAM3_E RRE	Interrupt redirect enable for SRAM parity error occurred in SRAM channel 3. For IXP2800 Rev B only. Otherwise Reserved.	RO, RW for IXP2xxx Rev B	0
[8]	SRAM2_E RRE	Interrupt redirect enable for SRAM parity error occurred in SRAM channel 2. For IXP28xx Rev B only. Otherwise Reserved.	RO, RW for IXP2xxx Rev B	0
[7]	SRAM1_E RRE	Interrupt redirect enable for SRAM parity error occurred in SRAM channel 1. For IXP2xxx Rev B only. Otherwise Reserved.	RO, RW for IXP2xxx Rev B	0
[6]	SRAM0_E RRE	Interrupt redirect enable for SRAM parity error occurred in SRAM channel 0. For IXP2xxx Rev B only. Otherwise Reserved.	RO, RW for IXP2xxx Rev B	0
[5]	DRAM2_E CC_MAJE	Interrupt redirect enable for Uncorrectable ECC error occurred in DRAM channel 2. For IXP2800 Rev B only. Otherwise Reserved.	RO, RW for IXP2xxx Rev B	0



Bits	Field	Description	RW	Reset
[27:26]	PIL	PCI_INTx_L: Reflects that state of the PCI interrupt pins, regardless of whether the IXP processor is driving the pin or not. bit 26: PCI_INTA_L bit 27: PCI_INTB_L	RO	0x0
[25]	PCII	Soft Interrupt from PCI to Intel XScale® core; Write 1 to PCI_Control[2] to set	RO	0x0
[24]	Reserved	Read as 0x0	RO	0x0
[23]	SP_FINT	Slow Port interrupt. This interrupt is forwarded from the external framer. To clear this bit, refer to the manual of the framer device. For IXP2xxx Rev B only. Otherwise Reserved.	RO	0
[22]	PMU_INT	PMU interrupt. To clear, write 0 to the PMU status register. For IXP2400 Rev B only. Otherwise Reserved.	RO	0
[21:18]	TIMER[3:0]_UFLW	Timer underflow indicator. This bit is set when timer has decremented to zero. Clear this bit by writing any value to the TimerClear register. For IXP2xxx Rev B only. Otherwise Reserved.	RO	0
[17]	GPIO_INT	Indicates an interrupt request from the GPIO unit. To clear, write 1 to the GPIO interrupt status register. For IXP2xxx Rev B only. Otherwise Reserved.	RO	0
[16]	UART_INT	Indicates an UART interrupt request. There are multiple conditions which triggers the interrupt, refer to the UART section to clear the interrupt. For IXP2xxx Rev B only. Otherwise Reserved.	RO	0
[15:12]	ME1_4..ME1_7	A 1 in a bit position indicates that the associated interrupt source is both active and enabled. To clear, write 1 to the appropriate IRQ_THD_RAW_STATUS_3 (\$= A or B) register. ME1_4 = Cluster 1, ME 4 ME1_7 = Cluster 1, ME 7 For IXP28xx Rev B only. Otherwise Reserved.	RO	0x0

Bits	Field	Description	RW	Reset
[28]	DMA1NBM	DMA channel 1 not busy	RW	0
[27:26]	PILM	PCI_INTA_L: Reflects that state of the PCI interrupt pins, regardless of whether the IXP processor is driving the pin or not. bit 26: PCI_INTA_L enable bit bit 27: PCI_INTB_L enable bit	RW	0
[25]	PCIIM	Mask the Soft Interrupt to Intel XScale® core from PCI_Control[2]	RW	0
[24]	Reserved	Read as 0x0	RO	0
[23:0]	XSCALE_INT_ENABLE	The individual enable bits for redirecting the Intel XScale® core interrupt as PCI_INTA_L. These bits correspond to bits [23:0] of the XSCALE_INT_STATUS register. For IXP2xxx Rev B only. Otherwise Reserved.	RO, RW for IXP2xxx Rev B	0x0

5.9.2.23 ME_PUSH_STATUS

This register is to signal the completion of the DMA channel to wake up the particular Microengine that started the DMA. A 1 in a bit position indicates that the associated channel is pushing the Microengine DMA Completion status.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
DMA3	DMA2	DMA1	RESERVED																											

Bits	Field	Description	RW	Reset
[31]	DMA3M	DMA channel 3.	RO	0
[30]	DMA2M	DMA channel 2. This bit is reserved for the IXP2800.	RO	0
[29]	DMA1M	DMA channel 1.	RO	0
[28:0]	RESERVED	Reserved. Read as 0x0	RO	0

5.9.2.24 ME_PUSH_ENABLE

This register is used to mask the Auto-Push source channels and define which active channels can push the status to the Microengine. A one in a particular bit location will enable that auto-push.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
DMA3	DMA2	DMA1	RESERVED																											



Bits	Field	Description	RW	Reset
[31]	DMA3M	DMA channel 3	RW	0
[30]	DMA2M	DMA channel 2, This bit is reserved for the IXP2800.	RW	0
[29]	DMA1M	DMA channel 1	RW	0
[28:0]	Reserved	Read as 0x0	RO	0

5.10 Intel XScale® Core Local CSRs

There are three classes of Intel XScale® core Local CSRs:

- Interrupt Controller ([Section 5.10.1](#))
- Hash Operation ([Section 5.10.2](#))
- Breakpoint ([Section 5.10.3](#))

5.10.1 Interrupt Controller (Intel XScale® Core)

Table 5-67 shows the offset addresses of the Interrupt Controller CSRs. Refer to [Chapter 4, “Address Maps”](#) for the base address and details on how they are accessed. These CSRs can only be accessed by the Intel XScale® core.

Note: For the thread related interrupt registers, substitute 0-8 for the # symbol, and A and B for the \$ symbol. Also the enable and enable set registers share the same register address.

Table 5-67. Intel XScale® Core Gasket Configuration Register Map

Abbreviation	Address [9:0]	Name	Description	
FIQ_RAW_STATUS	0x00	FIQ Raw Interrupt Status	FIQ un-masked interrupt status	Section 5.10.1.1
IRQ_RAW_STATUS	0x00	IRQ Raw Interrupt Status	IRQ un-masked interrupt status	Section 5.10.1.1
FIQ_STATUS	0x04	FIQ Interrupt Status	FIQ masked interrupt status	Section 5.10.1.2
IRQ_STATUS	0x08	IRQ Interrupt Status	IRQ masked interrupt status	Section 5.10.1.2
FIQ_ENABLE	0x0C	FIQ Enable	FIQ interrupt enable	Section 5.10.1.3
FIQ_ENABLE_SET	0x0C	FIQ Enable Set	FIQ interrupt enable set (W1S)	Section 5.10.1.4
IRQ_ENABLE	0x10	IRQ Enable	IRQ interrupt enable	Section 5.10.1.3
IRQ_ENABLE_SET	0x10	IRQ Enable Set	IRQ interrupt enable set (W1S)	Section 5.10.1.4
FIQ_ENABLE_CLR	0x14	FIQ Enable Clear	FIQ interrupt enable clear (W1C)	Section 5.10.1.5
IRQ_ENABLE_CLR	0x18	IRQ Enable Clear	IRQ interrupt enable clear (W1C)	Section 5.10.1.5
FIQ_ERR_RAW_STATUS	0x1C	FIQ Raw Error Status	FIQ un-masked interrupt status	Section 5.10.1.8
IRQ_ERR_RAW_STATUS	0x1C	IRQ Raw Error Status	IRQ un-masked interrupt status	Section 5.10.1.8
FIQ_ERR_STATUS	0x20	FIQ Error Interrupt Status	FIQ masked interrupt status	Section 5.10.1.9
IRQ_ERR_STATUS	0x24	IRQ Error Interrupt Status	IRQ masked interrupt status	Section 5.10.1.9
FIQ_ERR_ENABLE	0x28	FIQ Error Enable	FIQ error interrupt enable	Section 5.10.1.10
FIQ_ERR_ENABLE_SET	0x28	FIQ Error Enable Set	FIQ interrupt enable set (W1S)	Section 5.10.1.11

Table 5-67. Intel XScale® Core Gasket Configuration Register Map

Abbreviation	Address [9:0]	Name	Description	
IRQ_ERR_ENABLE	0x2C	IRQ Error Enable	IRQ error interrupt enable	Section 5.10.1.10
IRQ_ERR_ENABLE_SET	0x2C	IRQ Error Enable Set	Setting of the IRQ error enable register.	Section 5.10.1.11
FIQ_ERR_ENABLE_CLEAR	0x30	FIQ Error Enable Clear	Clearing of the FIQ error enable register.	Section 5.10.1.12
IRQ_ERR_ENABLE_CLEAR	0x34	IRQ Error Enable Clear	Clearing of the IRQ enable register.	Section 5.10.1.12
FIQ_RAW_ATTEN_STATUS	0x38	FIQ Raw Attention Status	FIQ un-masked version of the ME's attention interrupt request.	Section 5.10.1.13
IRQ_RAW_ATTEN_STATUS	0x38	IRQ Raw Attention Status	IRQ un-masked version of the ME's attention interrupt request.	Section 5.10.1.13
FIQ_ATTEN_STATUS	0x3C	FIQ Attention Status	FIQ masked version of the ME's attention interrupt request.	Section 5.10.1.14
IRQ_ATTEN_STATUS	0x40	IRQ Attention Status	IRQ masked version of the ME's attention interrupt request.	Section 5.10.1.14
FIQ_ATTEN_ENABLE	0x44	FIQ Attention Enable	FIQ attention request enable	Section 5.10.1.15
FIQ_ATTEN_ENABLE_SET	0x44	FIQ Attention Enable Set	FIQ attention request enable set (W1S)	Section 5.10.1.16
IRQ_ATTEN_ENABLE	0x48	IRQ Attention Enable	IRQ attention request enable	Section 5.10.1.15
IRQ_ATTEN_ENABLE_SET	0x48	IRQ Attention Enable Set	IRQ attention request enable set (W1S)	Section 5.10.1.16
FIQ_ATTEN_ENABLE_CLEAR	0x4C	FIQ Attention Enable Clear	Clearing the FIQ attention enable	Section 5.10.1.17
IRQ_ATTEN_ENABLE_CLEAR	0x50	IRQ Attention Enable Clear	Clearing the IRQ attention enable.	Section 5.10.1.17
SOFT_INT	0x54	FIQ Soft Interrupt	Software triggered FIQ interrupt	Section 5.10.1.6
SCRATCH_RING_STATUS	0x58	FIQ Raw Ring Full Status	FIQ un-masked version of the Shac's ring full indicator	Section 5.10.1.7

Table 5-67. Intel XScale® Core Gasket Configuration Register Map

Abbreviation	Address [9:0]	Name	Description	
IRQ_ERR_ENABLE	0x2C	IRQ Error Enable	IRQ error interrupt enable	Section 5.10.1.10
IRQ_ERR_ENABLE_SET	0x2C	IRQ Error Enable Set	Setting of the IRQ error enable register.	Section 5.10.1.11
FIQ_ERR_ENABLE_CLEAR	0x30	FIQ Error Enable Clear	Clearing of the FIQ error enable register.	Section 5.10.1.12
IRQ_ERR_ENABLE_CLEAR	0x34	IRQ Error Enable Clear	Clearing of the IRQ enable register.	Section 5.10.1.12
FIQ_RAW_ATTEN_STATUS	0x38	FIQ Raw Attention Status	FIQ un-masked version of the ME's attention interrupt request.	Section 5.10.1.13
IRQ_RAW_ATTEN_STATUS	0x38	IRQ Raw Attention Status	IRQ un-masked version of the ME's attention interrupt request.	Section 5.10.1.13
FIQ_ATTEN_STATUS	0x3C	FIQ Attention Status	FIQ masked version of the ME's attention interrupt request.	Section 5.10.1.14
IRQ_ATTEN_STATUS	0x40	IRQ Attention Status	IRQ masked version of the ME's attention interrupt request.	Section 5.10.1.14
FIQ_ATTEN_ENABLE	0x44	FIQ Attention Enable	FIQ attention request enable	Section 5.10.1.15
FIQ_ATTEN_ENABLE_SET	0x44	FIQ Attention Enable Set	FIQ attention request enable set (W1S)	Section 5.10.1.16
IRQ_ATTEN_ENABLE	0x48	IRQ Attention Enable	IRQ attention request enable	Section 5.10.1.15
IRQ_ATTEN_ENABLE_SET	0x48	IRQ Attention Enable Set	IRQ attention request enable set (W1S)	Section 5.10.1.16
FIQ_ATTEN_ENABLE_CLEAR	0x4C	FIQ Attention Enable Clear	Clearing the FIQ attention enable	Section 5.10.1.17
IRQ_ATTEN_ENABLE_CLEAR	0x50	IRQ Attention Enable Clear	Clearing the IRQ attention enable.	Section 5.10.1.17
SOFT_INT	0x54	FIQ Soft Interrupt	Software triggered FIQ interrupt	Section 5.10.1.6
SCRATCH_RING_STATUS	0x58	FIQ Raw Ring Full Status	FIQ un-masked version of the Shac's ring full indicator	Section 5.10.1.7

Table 5-67. Intel XScale® Core Gasket Configuration Register Map

Abbreviation	Address [9:0]	Name	Description	
FIQ_THD_RAW_STATUS_A_#	0x60-0x6C	FIQ Thread A Raw Status	FIQ un-masked thread A interrupt status 0x60 and 0x68 for the IXP2400 only	Section 5.10.1.18
IRQ_THD_RAW_STATUS_A_#	0x60-0x6C	IRQ Thread A Raw Status	IRQ un-masked thread A interrupt status. 0x60 and 0x68 for the IXP2400 only	
FIQ_THD_RAW_STATUS_B_#	0x80-0x8C	FIQ Thread B Raw Status	FIQ un-masked thread B interrupt status. 0x80 and 0x88 for the IXP2400 only	
IRQ_THD_RAW_STATUS_B_#	0x80-0x8C	IRQ Thread B Raw Status	IRQ un-masked thread B interrupt status. 0x80 and 0x88 for the IXP2400 only	
FIQ_THD_STATUS_A_#	0xA0-0xAC	FIQ Thread A Status	FIQ masked thread A interrupt status 0xA0 and 0xA8 for the IXP2400 only	Section 5.10.1.19
FIQ_THD_STATUS_B_#	0xC0-0xCC	FIQ Thread B Status	FIQ masked thread B interrupt status 0xC0 and 0xC8 for the IXP2400 only	
IRQ_THD_STATUS_A_#	0xE0-0xEC	IRQ Thread A Status	IRQ masked thread A interrupt status 0xE0 and 0xE8 for the IXP2400 only	
IRQ_THD_STATUS_B_#	0x100-0x10C	IRQ Thread B Status	IRQ masked thread B interrupt status 0x100 and 0x108 for the IXP2400 only	
FIQ_THD_ENABLE_A_#	0x120-0x12C	FIQ Thread A Enable	FIQ thread A interrupt enable 0x120 and 0x128 for the IXP2400 only	Section 5.10.1.20
FIQ_THD_ENABLE_SET_A_#	0x120-0x12C	FIQ Thread A Enable Set	Setting of the FIQ thread A interrupt enable 0x120 and 0x128 for the IXP2400 only	Section 5.10.1.21
FIQ_THD_ENABLE_B_#	0x140-0x14C	FIQ Thread B Enable	FIQ thread B interrupt enable 0x140 and 0x148 for the IXP2400 only	Section 5.10.1.20
FIQ_THD_ENABLE_SET_B_#	0x140-0x14C	FIQ Thread B Enable Set	Setting of the FIQ thread B interrupt enable 0x140 and 0x148 for the IXP2400 only	Section 5.10.1.21
IRQ_THD_ENABLE_A_#	0x160-0x16C	IRQ Thread A Enable	IRQ thread A interrupt enable 0x160 and 0x168 for the IXP2400 only	Section 5.10.1.20

Table 5-67. Intel XScale® Core Gasket Configuration Register Map

Abbreviation	Address [9:0]	Name	Description	
IRQ_THD_ENABLE_SET_A_#	0x160-0x16C	IRQ Thread A Enable Set	Setting of the IRQ thread A interrupt enable 0x160 and 0x168 for the IXP2400 only	Section 5.10.1.21
IRQ_THD_ENABLE_B_#	0x180-0x18C	IRQ Thread B Enable	IRQ thread B interrupt enable 0x180 and 0x188 for the IXP2400 only	Section 5.10.1.20
IRQ_THD_ENABLE_SET_B_#	0x180-0x18C	IRQ Thread B Enable Set	Setting of the IRQ thread B interrupt enable 0x180 and 0x188 for the IXP2400 only	Section 5.10.1.21
FIQ_THREAD_ENABLE_CLEAR_A_#	0x1A0-0x1AC	FIQ Thread A Enable Clear	Clearing of the FIQ thread A interrupt enable 0x1A0 and 0x1A8 for the IXP2400 only	Section 5.10.1.22
FIQ_THREAD_ENABLE_CLEAR_B_#	0x1C0-0x1CC	FIQ Thread B Enable Clear	Clearing of the FIQ thread B interrupt enable 0x1C0 and 0x1C8 for the IXP2400 only	
IRQ_THREAD_ENABLE_CLEAR_A_#	0x1E0-0x1EC	IRQ Thread A Enable Clear	Clearing of the IRQ thread A interrupt enable 0x1E0 and 0x1E8 for the IXP2400 only	
IRQ_THREAD_ENABLE_CLEAR_B_#	0x200-0x20C	IRQ Thread B Enable Clear	Clearing of the IRQ thread B interrupt enable. 0x200 and 0x208 for the IXP2400 only	

5.10.1.1 {IRQ,FIQ}RAW_STATUS

The {IRQ,FIQ}RAW_STATUS register always contain identical data.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0				
RESERVE D				THD96-127_B	THD64-95_B	THD32-63_B	THD0-31_B	RESERVE D				THD96-127_A	THD64-95_A	THD32-63_A	THD0-31_A	PCI_INT	ME_ATT	PCI_DBELL	DMA2_DONE	DMA1_DONE	DMA0_DONE	SP_FINT	PMU_INT	TMR[3:0]_UFLW			GPI0_INT	UART_INT	ERR_SUM	SOFTINT

Bits	Field	Description	RW	Reset
[31:28]	RESERVED	Reserved	RO	0
[27]	THD96-127_B	OR of all interrupt bits in the THD_RAW_STATUS_B_3 register. To clear, write 1 to the THD_RAW_STATUS_B_3 register. IXP2800 only. Reserved on IXP2400.	RO	0
[26]	THD64-95_B	OR of all interrupt bits in the THD_RAW_STATUS_B_2 register. To clear, write 1 to the THD_RAW_STATUS_B_2 register.	RO	0
[25]	THD32-63_B	OR of all interrupt bits in the THD_RAW_STATUS_B_1 register. To clear, write 1 to the THD_RAW_STATUS_B_1 register. IXP2800 only. Reserved on IXP2400.	RO	0
[24]	TH0-31_B	OR of all interrupt bits in the THD_RAW_STATUS_B_0 register. To clear, write 1 to the THD_RAW_STATUS_B_0 register.	RO	0
[23:20]	RESERVED	Reserved	RO	0
[19]	THD96-127_A	OR of all interrupt bits in the THD_RAW_STATUS_A_3 register. To clear, write 1 to the THD_RAW_STATUS_A_3 register. IXP2800 only. Reserved on IXP2400.	RO	0
[18]	THD64-95_A	OR of all interrupt bits in the THD_RAW_STATUS_A_2 register. To clear, write 1 to the THD_RAW_STATUS_A_2 register.	RO	0
[17]	THD32-63_A	OR of all interrupt bits in the THD_RAW_STATUS_A_1 register. To clear, write 1 to the THD_RAW_STATUS_A_1 register. IXP2800 only. Reserved on IXP2400.	RO	0
[16]	THD0-31_A	OR of all interrupt bits in the THD_RAW_STATUS_A_0 register. To clear, write 1 to the THD_RAW_STATUS_A_0 register.	RO	0
[15]	PCI_INT	The OR of external PCI interrupt A & B.	RO	0
[14]	ME_ATTEN	OR of all the bits in the Microengine attention register.	RO	0
[13]	PCI_DOORBELL	A PCI device has set the doorbell interrupt. To Clear, write 1 to the PCI's Intel XScale® core doorbell register.	RO	0
[12]	DMA2_DONE	Completion status from the DMA2 engine.	RO	0
[11]	DMA1_DONE	Completion status from the DMA1 engine. This bit is reserved for the IXP2800.	RO	0
[10]	DMA0_DONE	Completion status from the DMA0 engine.	RO	0
[9]	SP_FINT	Slow Port interrupt. This interrupt is forwarded from the external framer. To clear this bit, refer to the manual of the framer device.	RO	0
[8]	PMU_INT	PMU interrupt. To clear, write 0 to the PMU status register.	RO	0
[7:4]	TIMER[3:0]_UFLW	Timer underflow indicator. This bit is set when timer has decremented to zero. Clear this bit by writing any value to the TimerClear register.	RO	0
[3]	GPIO_INT	Indicates an interrupt request from the GPIO unit. To clear, write 1 to the GPIO interrupt status register.	RO	0

Bits	Field	Description	RW	Reset
[2]	UART_INT	Indicates an UART interrupt request. There are multiple conditions which triggers the interrupt, refer to the UART section to clear the interrupt.	RO	0
[1]	ERROR_SUM	OR of all interrupt bits in the ErrorStatus register.	RO	0
[0]	SOFTINT	Software is able to generate IRQ or FIQ through this bit. To set this bit, write 1 to the {IRQ,FIQ}SOFT[0]. To clear this bit, write 0 to the [IRQ,FIQ]SOFT[0].	RO	0

5.10.1.2 {IRQ,FIQ}STATUS

This read-only register is a bit-wise AND of {IRQ,FIQ}RAW_STATUS and {IRQ,FIQ}ENABLE.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
{IRQ,FIQ}STATUS ref to Section 5.10.1.1 for bit layout																														

Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ}STATUS	A 1 in a bit position indicates that the associated interrupt source is both active and enabled. The IRQ/FIQ will be asserted to the processor based on the OR of all of the bits in this register.	RO	0

5.10.1.3 {IRQ,FIQ}ENABLE

This read-only register is used to mask the interrupt input sources and defines which active sources generate an interrupt request to the Intel XScale® core. The value of this register can only be changed by writing to the {IRQ,FIQ}ENABLE_SET and {IRQ,FIQ}ENABLE_CLR registers.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
{IRQ,FIQ}ENABLES ref to Section 5.10.1.1 for bit layout																															

Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ} ENABLE	A 1 indicates that the associated interrupt source is enabled and allows an interrupt request. A 0 indicates that the interrupt is disabled.	RO	0

5.10.1.4 {IRQ,FIQ}ENABLE_SET

This write-only register is used to set bits in the {IRQ,FIQ}ENABLE register.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
{IRQ,FIQ}ENABLE_SET ref to Section 5.10.1.1 for bit layout																															

Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ} ENABLE_SET	When writing to this location, each data bit that is high causes the corresponding bit in the {IRQ,FIQ}ENABLE register to be set. Data bits that are low have no effect.	W1S	0

5.10.1.5 {IRQ,FIQ}ENABLE_CLR

This write-only register is used to clear bits in the {IRQ,FIQ}ENABLE register.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
{IRQ,FIQ}ENABLE_CLR ref to Section 5.10.1.1 for bit layout																															

Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ} ENABLE_CLR	When writing to this location, each data bit that is high causes the corresponding bit in the {IRQ,FIQ}ENABLE register to be cleared. Data bits that are low have no effect.	W1C	0

5.10.1.6 {IRQ,FIQ}SOFT_INT

This write-only register can be used to generate an IRQ/FIQ under software control.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																															SFT_INT	

Bits	Field	Description	RW	Reset
[31:1]	RESERVED	Reserved. Don't care	RO	0
[0]	{IRQ,FIQ} SOFT_INT	This bit generates bit[0] (as either 0 or 1) of the RAW_STATUS register (and its current state can be found by reading that register).	W	0

5.10.1.7 SCRATCH_RING_STATUS

This read-only register of the scratch ring full status. A '1' in a particular bit location indicates the corresponding scratch ring (Scratch rings are numbered from 0 to 15) is full. A zero indicate the ring is not full and the Intel XScale® core can issue a put into the ring.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																RINGFULL																

Bits	Field	Description	RW	Reset
[31:16]	RESERVED	Reserved	RO	0
[15:0]	RINGFULL	Scratch ring full indicator. A one indicate when the Scratch memory ring is full. A zero indicates that it's safe to issue put to the ring.	RO	0

5.10.1.8 {IRQ,FIQ}ERR_RAW_STATUS

This read-only register is an unmasked version of the error status. The {IRQ,FIQ}ERR_RAW_STATUS read-only register always contain identical data.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED				SP_INT	PCI_ERR	MEDIA_ERR	RESERVED				SRAM3_ERR	SRAM2_ERR	SRAM1_ERR	SRAM0_ERR	RESERVED				DRAM2_ECC_MIN	DRAM2_ECC_MAX	DRAM1_ECC_MIN	DRAM1_ECC_MAX	DRAM0_ECC_MIN	DRAM0_ECC_MAX				

Bits	Field	Description	RW	Reset
[31:27]	RESERVED	Reserved	RO	0
[26]	SP_INT	Slow Port interrupt. To clear, write 1 to the Slow Port's fault status register.	RO	0x0
[25]	PCI_ERR	PCI error indicator. To clear, read the XSCALE_ERR_STATUS register in the PCI unit to identify the source of the interrupt and then write 1 to the corresponding bit in the PCI_CONTROL register. Note that some interrupt sources have to be cleared by writing 1 to the corresponding bits in the PCI_CONTROL register and the PCI_CMD_STAT register. for details on the requirements of clearing each interrupt refer to the bit description in the XSCALE_ERR_STATUS register. in the PCI unit.	RO	0
[24]	MEDIA_ERR	Media error indicator. To clear, write 1 to the error bit in the MSF interrupt status register.	RO	0
[23:20]	RESERVED	Reserved	RO	0
[19]	SRAM3_ERR	Indicates a SRAM parity error has occurred in SRAM channel 3. To clear, write 1 to the error bit in the SRAM3 parity status register. IXP2800 only. Reserved on IXP2400.	RO	0
[18]	SRAM2_ERR	Indicates a SRAM parity error has occurred in SRAM channel 2. To clear, write 1 to the error bit in the SRAM2 parity status register. IXP2800 only. Reserved on IXP2400.	RO	0
[17]	SRAM1_ERR	Indicates a SRAM parity error has occurred in SRAM channel 1. To clear, write 1 to the error bit in the SRAM1 parity status register.	RO	0

Bits	Field	Description	RW	Reset
[16]	SRAM0_ERR	Indicates a SRAM parity error has occurred in SRAM channel 0. To clear, write 1 to the error bit in the SRAM0 parity status register.	RO	0
[15:6]	RESERVED	Reserved	RO	0
[5]	DRAM2_ECC_MAJ	Uncorrectable ECC error occurred in DRAM channel 3. To clear, write 1 to the error bit in the DRAM3 error status register IXP2800 only. Reserved on IXP2400.	RO	0
[4]	DRAM2_ECC_MIN	Correctable ECC error occurred in DRAM channel 3. To clear, write 1 to the error bit in the DRAM3 error status register. IXP2800 only. Reserved on IXP2400.	RO	0
[3]	DRAM1_ECC_MAJ	Uncorrectable ECC error occurred in DRAM channel 2. To clear, write 1 to the error bit in the DRAM2 error status register IXP2800 only. Reserved on IXP2400.	RO	0
[2]	DRAM1_ECC_MIN	Correctable ECC error occurred in DRAM channel 1. To clear, write 1 to the error bit in the DRAM1 error status register. IXP2800 only. Reserved on IXP2400.	RO	0
[1]	DRAM0_ECC_MAJ	Uncorrectable ECC error occurred in DRAM channel 0. To clear, write 1 to the error bit in the DRAM0 error status register	RO	0
[0]	DRAM0_ECC_MIN	Correctable ECC error occurred in DRAM channel 0. To clear, write 1 to the error bit in the DRAM0 error status register.	RO	0

Note: For the IXP2400, there are only two SRAM channels and one DRAM channel, therefore bits 19, 18, and 5-2 are reserved bits.

5.10.1.9 {IRQ,FIQ}ERR_STATUS

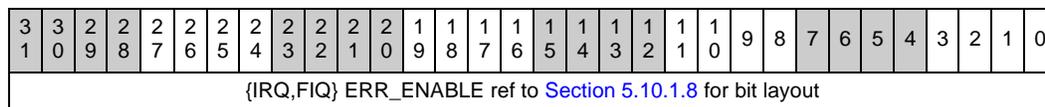
This read-only register is a bit-wise AND of {IRQ,FIQ}ERR_RAW_STATUS and {IRQ,FIQ}ERR_ENABLE.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
{IRQ,FIQ} ERR_STATUS ref to Section 5.10.1.8 for bit layout																															

Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ}ERR_STATUS	A 1 in a bit position indicates that the associated interrupt source is both active and enabled. The IRQ/FIQ will be asserted to the processor based on the OR of all of the bits in this register if the error bit in {IRQ,FIQ}ENABLE is set.	RO	0

5.10.1.10 {IRQ,FIQ}ERR_ENABLE

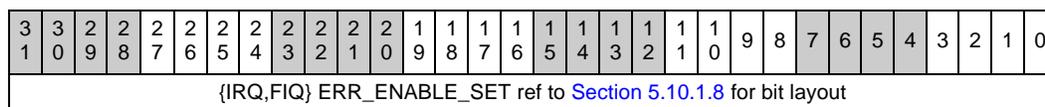
This read-only register is used to mask the error interrupt sources and defines which active errors can generate an interrupt request to the Intel XScale® core. The value of this register can only be changed by writing to the {IRQ,FIQ}ERR_ENABLE_SET and {IRQ,FIQ}ERR_ENABLE_CLR registers.



Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ} ERR_ENABLE	A 1 indicates that the associated error interrupt is enabled and allows an interrupt request. A 0 indicates that the interrupt is disabled.	RO	0

5.10.1.11 {IRQ,FIQ}ERR_ENABLE_SET

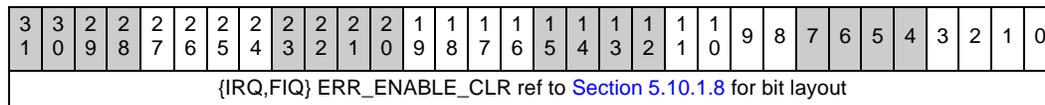
This write-only register is used to set bits in the {IRQ,FIQ}ERR_ENABLE register.



Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ} ERR_ENABLE_SET	When writing to this location, each data bit that is high causes the corresponding bit in the {IRQ,FIQ}ERR_ENABLE register to be set. Data bits that are low have no effect.	W1 S	0

5.10.1.12 {IRQ,FIQ}ERR_ENABLE_CLR

This write-only register is used to clear bits in the {IRQ,FIQ}ERR_ENABLE register.



Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ} ERR_ENABLE_CLR	When writing to this location, each data bit that is high causes the corresponding bit in the {IRQ,FIQ}ERR_ENABLE register to be cleared. Data bits that are low have no effect.	W1 C	0

Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ} THD_RAW_STATUS	Threads 0-31 - THD_RAW_STATUS_A_0 Threads 32-63 - THD_RAW_STATUS_A_1 Threads 64-95 - THD_RAW_STATUS_A_2 Threads 96-127 - THD_RAW_STATUS_A_3 Threads 0-31 - THD_RAW_STATUS_B_0 Threads 32-63 - THD_RAW_STATUS_B_1 Threads 64-95 - THD_RAW_STATUS_B_2 Threads 96-127 - THD_RAW_STATUS_B_3 A 1 in a bit position indicates that the associated thread initiated an interrupt For IXP2400, only A_0, A_2, B_0 and B_2 registers are valid.	RW1C	0

5.10.1.19 {IRQ,FIQ}THD_STATUS_\$_# (\$= A, B and # = 0 - 3)

This read-only register is a bit-wise AND of {IRQ,FIQ}THD_RAW_STATUS_\$_# and {IRQ,FIQ}THREADENABLE_\$_#.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
{IRQ,FIQ}THREADSTATUS																															

Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ} THD_STATUS	A 1 in a bit position indicates that the associated thread interrupt is both active and enabled. The IRQ/FIQ will be asserted to the processor based on the OR of all of the bits in this register if the error bit in {IRQ,FIQ}ENABLE is set. For IXP2400, # must be 0 or 2.	RO	0

5.10.1.20 {IRQ,FIQ}THD_ENABLE_\$_# (\$= A, B and # = 0 - 3)

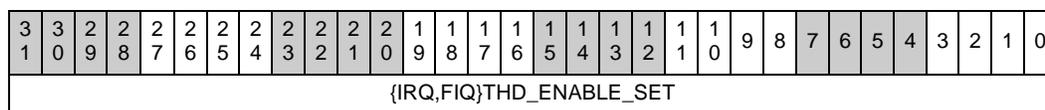
This read-only register is used to mask the thread interrupt sources and defines which active thread interrupts can generate an interrupt request to the Intel XScale® core. The value of this register can only be changed by writing to the {IRQ,FIQ}THD_ENABLE_SET_\$_# and {IRQ,FIQ}THD_ENABLE_CLR_\$_# registers.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
{IRQ,FIQ}THD_ENABLE																															

Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ}THREAD ENABLE	A 1 indicates that the associated thread interrupt is enabled and allows an interrupt request. A 0 indicates that the interrupt is disabled. For IXP2400, # must be 0 or 2.	RO	0

5.10.1.21 {IRQ,FIQ}THD_ENABLE_SET_\$_# (\$= A, B and # = 0 - 3)

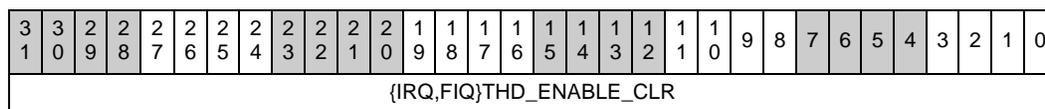
This write-only register is used to set bits in the {IRQ,FIQ}THD_ENABLE_\$_# register.



Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ}THREAD ENABLESET	When writing to this location, each data bit that is high causes the corresponding bit in the {IRQ,FIQ}THREADENABLE register to be set. Data bits that are low have no effect. For IXP2400, # must be 0 or 2.	W1S	0

5.10.1.22 {IRQ,FIQ}THD_ENABLE_CLR_\$_# (\$= A, B and # = 0 - 3)

This write-only register is used to clear bits in the {IRQ,FIQ}THD_ENABLE register.



Bits	Field	Description	RW	Reset
[31:0]	{IRQ,FIQ}THD_ ENABLE_CLR	When writing to this location, each data bit that is high causes the corresponding bit in the {IRQ,FIQ}THD_ENABLE register to be cleared. Data bits that are low have no effect. For IXP2400, # must be 0 or 2.	W1C	0

5.10.2 Hash Operation (Intel XScale® Core)

Table 5-68 lists the Hash Unit Registers that are addressed by the Intel XScale® core when performing has operations. The registers are designed as pairs of operand and result registers. The Operand registers are write-only and the Results registers are read-only and they use the same address. Refer to Chapter 4, “Address Maps” for the base address and details on how they are accessed. These CSRs can only be accessed by the Intel XScale® core.

Table 5-68. Hash Operation/Result Register Map (Sheet 1 of 2)

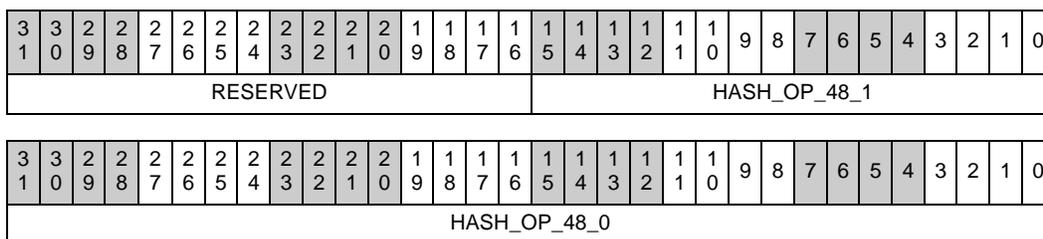
Abbreviation	Address	Description	Section
HASH_OP_48_0	0x00	Hash operand and result registers for 48 bit hashing	Section 5.10.2.1
HASH_OP_48_1	0x04		

Table 5-68. Hash Operation/Result Register Map (Sheet 2 of 2)

Abbreviation	Address	Description	Section
HASH_OP_64_0	0x10	Hash operand and result registers for 64 bit hashing	Section 5.10.2.2
HASH_OP_64_1	0x14		
HASH_OP_128_0	0x20	Hash operand and result registers for 128 bit hashing	Section 5.10.2.3
HASH_OP_128_1	0x24		
HASH_OP_128_2	0x28		
HASH_OP_128_3	0x2C		
HASH_DONE	0x30	Hash operation complete status register	Section 5.10.2.4

5.10.2.1 HASH_OP_48_# (# = 0,1)

These registers contain the operands and results for 48-bit hash keys. HASH_OP_48_1 contains the most significant 16 bits of the 48-bit hash operand/result. HASH_OP_48_0 contains the least significant 32 bits of the 48-bit hash operand/result.



Bits	Field	Description	RW	Reset
[31:16]	Reserved		RO	undef
[15:0]	HASH_OP_48_1	The most significant 16 bits of the 48-bit hash operand/result.	RW	undef

Bits	Field	Description	RW	Reset
[31:0]	HASH_OP_48_0	The least significant 32 bits of the 48-bit hash operand/result.	RW	undef

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
HASH_OP_128_0																															

Bits	Field	Description	RW	Reset
[31:0]	HASH_OP_128_3	The most significant 32 bits (127 to 96) of the 128-bit hash operand/result.	RW	undef

Bits	Field	Description	RW	Reset
[31:0]	HASH_OP_128_2	Contains bits 64 to 95 of the 128-bit hash operand/result.	RW	undef

Bits	Field	Description	RW	Reset
[31:0]	HASH_OP_128_1	Contains bits 32 to 63 of the 128-bit hash operand/result.	RW	undef

Bits	Field	Description	RW	Reset
[31:0]	HASH_OP_128_0	The least significant 32 bits (31 to 0) of the 128-bit hash operand/result.	RW	undef

5.10.2.4 HASH_DONE

This register contain the flag which indicates when a Hash Operation has completed. It is cleared when the hash operands are written to HASH_OP by the Intel XScale® core, and set when the result is written to HASH_OP by the Hash Unit.

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
RESERVED																															DONE

Bits	Field	Description	RW	Reset
[31:1]	RESERVED	Reserved	RO	0
[0]	DONE	Hash result valid flag. 0—Hash result not yet valid. 1—Hash result is valid.	RO	0

5.10.3 Breakpoint (Intel XScale® Core)

This section provides details of a breakpoint feature within the Intel XScale® core that allows multiple sources for the Intel XScale® core break signal and allows the break signal to optionally be used as a Stop Clock source for the JTAG debug scan feature.

There is a second level of breakpoint enables, similar to the second level of enables for the IRQ and FIQ interrupts. The first level of enables is shared with the FIQ. FIQ and IRQ can remain enabled while the breakpoint feature is enabled. A control bit is provided that allows the break signal to be used either to breakpoint the Intel XScale® core or to stop the core clocks. When used to stop the core clocks, the status of this signal can be read out via the JTAG port, and the JTAG debug scan feature used to read the internal state of the IXP2800/IXP2400 (excluding the Intel XScale® core).

The JTAG debug scan feature is a JTAG method for stopping the IXP2800/IXP2400 internal core clocks and reading out the state of IXP2800/IXP2400 to aid in determining the cause of a failure. The additional break sources controlled by the Intel XScale® core allow the clocks to be stopped close to an event that can be specified. The additional Stop Clock sources increase the flexibility in stopping the clocks. They also increase the likelihood of capturing the state information that will help in determining the cause of a failure, where the failure state is transient.

Stopping the clocks is not a recoverable operation. Once the clocks are stopped, normal operation cannot be resumed without reset.

The breakpoint sources are derived from the first level of FIQ interrupt sources within the Intel XScale® core. All of the FIQ interrupt sources, except for the Thread Interrupts, can be used to either breakpoint the Intel XScale® core or stop the core clocks. Details are given in the register descriptions.

Figure 5-3 illustrates the signal flow through the two enable levels for IRQ, FIQ and the break signal. The first level of interrupt enables is shared with FIQ. The second level of interrupt enables is separate from FIQ.

Figure 5-3. Breakpoint Implementation

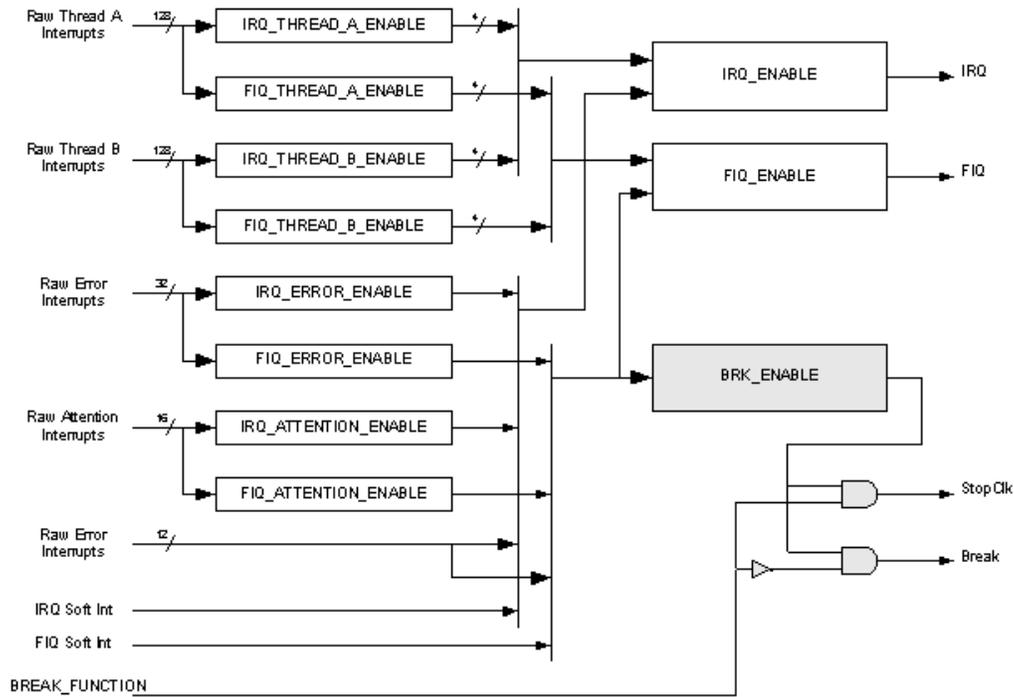


Table 5-69 lists the Interrupt Controller Registers. For the Intel XScale® core accesses, they are offsets from base address 0xd700 0000.

Table 5-69. Break Point Register Map for the Intel XScale® Core

Abbreviation	Address	Description	Section
BRK_RAW_STATUS	0x220	Breakpoint Raw Status Un-masked breakpoint status	Section 5.10.2.1
BRK_STATUS	0x224	Breakpoint Status Masked breakpoint status	
BRK_ENABLE	0x228	Breakpoint Enable	Section 5.10.2.2
BRK_ENABLE_SET	0x228	Breakpoint Enable Set	
BRK_ENABLE_CLEAR	0x22C	Breakpoint Enable Clear	Section 5.10.2.3

5.10.3.1 BRK_RAW_STATUS

The bits in this register indicate the raw values of the various breakpoint sources, without being masked. The lower 16-bit of this CSR always contains identical data to lower 16-bit of the {IRQ,FIQ}RawStatus CSRs.

This register currently does not include the PMU interrupts because they have not been defined yet for the FIQ and IRQ. Once defined for IRQ and FIQ, they will be included for the breakpoint in the identical form used for the interrupts.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
RESERVED																PCI_INT	ME_ATT	PCI_DOORBELL	DMA[2:0]_DONE	SP_FINT	PMU_INT	TIMER[3:0]_UFLOW			GPIO_INT	UART_INT	ERROR_SUM	SOFTINT			

Bits	Field	Description	RW	Reset
[31:16]	RESERVED	Reserved	RO	0
[15]	PCI_INT	The OR of external PCI interrupt A & B.	RO	0
[14]	ME_ATT	OR of all the bits in the ME attention register.	RO	0
[13]	PCI_DOORBELL	A PCI device has set the doorbell interrupt. To Clear, write 1 to the PCI's Intel XScale® core doorbell register.	RO	0
[12:10]	DMA[2:0]_DONE	Completion status from the DMA engine.	RO	0
[9]	SP_FINT	Slow Port framer interrupt. This interrupt is forwarded from the external framer. To clear this bit, refer to the manual of the framer device.	RO	0
[8]	PMU_INT	PMU interrupt. To clear, write 0 to the PMU interrupt status register.	RO	0
[7:4]	TIMER[3:0]_UFLOW	Timer underflow indicator. This bit is set when timer has decremented to zero. Clear this bit by writing any value to the TimerClear register.	RO	0
[3]	GPIO_INT	Indicates an interrupt request from the GPIO unit. To clear, write 1 to the GPIO interrupt status register.	RO	0
[2]	UART_INT	Indicates an UART interrupt request. There are multiple conditions which triggers the interrupt, refer to the UART section to clear the interrupt.	RO	0
[1]	ERROR_SUM	OR of all interrupt bits in the ErrorStatus register.	RO	0
[0]	SOFTINT	Software is able to generate a break through this bit. To set this bit, write 1 to the FIQ Soft[0]. To clear this bit, write 0 to the FIQ Soft[0].	RO	0

5.10.3.2 BRK_STATUS

This read-only register is a bit-wise AND of BRK_RAW_STATUS and BRK_ENABLE. This register is only useful when the breakpoint sources are used to generate an Intel XScale® core break. If used to stop the clocks, this register cannot be read after the clocks have been stopped. In this case, the source that stopped the clocks will be determined by state information that is scanned out via the JTAG port.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
BRK_STATUS																															

Bits	Field	Description	RW	Reset
[31:0]	BRK_STATUS	A 1 in a bit position indicates that the associated breakpoint source is both active and enabled. The breakpoint signal will be asserted based on the OR of all of the bits in this register.	RO	0

5.10.3.3 BRK_ENABLE

This read-only register is used to mask the breakpoint input sources and defines which active sources generate the breakpoint signal. The value of this register can only be changed by writing to the BRK_ENABLE_SET and BRK_ENABLE_CLR registers.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
BRK_ENABLE																															

Bits	Field	Description	RW	Reset
[31:0]	BRK_ENABLE	A 1 indicates that the associated breakpoint source is enabled and allows the breakpoint signal to become active. A 0 indicates that the breakpoint source is disabled.	RO	0

5.10.3.4 BRK_ENABLE_SET

This write-only register is used to set bits in the BRK_ENABLE register.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
BRK_ENABLE_SET																															

Bits	Field	Description	RW	Reset
[31:0]	BRK_ENABLE_SET	When writing to this location, each data bit that is high causes the corresponding bit in the BRK_ENABLE register to be set. Data bits that are low have no effect.	W1S	0

5.10.3.5 BRK_ENABLE_CLR

This write-only register is used to clear bits in the BRK_ENABLE register.

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0								
BRK_ENABLE_CLR																													

Bits	Field	Description	RW	Reset
[31:0]	BRK_ENABLE_CLR	When writing to this location, each data bit that is high causes the corresponding bit in the BRK_ENABLE register to be cleared. Data bits that are low have no effect.	W1C	0

5.11 Intel XScale® Co-Processors

The Intel XScale® Co-Processor register documentation is published as part of the Intel XScale® core Technology documents. This information is not duplicated here.

5.12 MSF differences between IXP2400 and IXP2800

This section describes the differences between IXP2400 and IXP2800 in terms of configuring and programming the Media and Switch Fabric Interface unit (MSF).

- On IXP2400, the RBUF elements have to be put into the RBUF element freelist before they can be used, after a reset. This is done by writing RBUF_ELEMENT_DONE with all element numbers. This is not required for IXP2800, which comes out of reset with all RBUF elements ready for use.
- In IXP2400, the TCW to TBUF element alignment varies depending on the TBUF size, whereas in IXP2800, it does not.

The indexing, in terms of (mpacket offset):(TCW offset)

IXP2400:

64 = 0:0, 1:1, 2:2

128 = 0:0, 2:1, 4:2

256 = 0:0, 4:1, 8:2

e.g. to transmit the 3rd mpacket in TBUF when element size = 128, the offset for Tx_Validate is $2*8=16$, and the offset for TBUF data is $4*64=256$

IXP2800:

64 = 0:0, 1:1, 2:2

128 = 0:0, 2:2, 4:4

256 = 0:0, 4:4, 8:8

e.g. to transmit the 3rd mpacket in TBUF when element size = 128, the offset for Tx_Validate is $4*8=32$, and the offset for TBUF data is $4*64=256$

- The SRB_OVERRIDE register has different functionality between IXP2400 and IXP2800. In IXP2400, it only has an effect in full duplex mode. In IXP2800, it has an effect in both full duplex and simplex mode.
- IXP2400 requires that the receive enable and transmit enable bits to be set only after the user software completes configuring the MSF unit. These bits are Rx_En[3:0] in the MSF_Rx_Control register and Tx_En[3:0] in the MSF_Tx_Control register. In other words, these two registers need to be written twice during the user software initialization sequence, once to configure the MSF unit by writing to the rest of these registers while keeping the Rx_En and Tx_En bits at 0, and another time to enable the desired channels using the Rx_En and Tx_En bits. The IXP2800 does not have this requirement.
- When the FCIFIFO/FCEFIFO is configured in Full-duplex mode, the IXP2800 automatically transmits idle CFrames over FCIFIFO/FCEFIFO. On the other hand, the IXP2400 does not automatically transmit idle CFrames in this case. Moreover, for IXP2400 in this case, the upper 2 ready bits of the CFrames transmitted over FCIFIFO/FCEFIFO should be ignored.
- In the IXP2800, an RBUF_OVERFLOW interrupt indicates lack of any buffer space to accept an incoming SPI-4 Frame or CFrame, and data is discarded. In the IXP2400, an RBUF_OVERFLOW interrupt indicates lack of an available RBUF element, and data is not discarded.
- In the IXP2800, the vertical parity must be counted for when writing the FCEFIFO register. The trailing bytes of the last CWord must be set to zero by the software. If the vertical parity does not fit in the last CWord written into the FIFO, an extra CWord equal to zero must be written as a place holder for the vertical parity. For example, writing a 5 byte data (d) CFrame with two bytes of header (h), 4 bytes of extension header (e) and 2 bytes of vertical parity (v) should have the following format: hhhh eeee, eeee dddd, dddd dd00, 0000 vvvv, where the comma separates the CWords written to this register, v is a place holder for the vertical parity, and 0 is padding.

Note: *The following differences result from UTOPIA/POS-PHY vs. SPI-4 differences.*

- IXP2800 implements SPI-4; IXP2400 implements UTOPIA Levels 1/2/3 and POS-PHY Levels 2/3. The CSIX implementation on both IXP2400 and IXP2800 should appear identical from a programmer's viewpoint.
- IXP2800 allows SPI-4 and CSIX modes to coexist; in IXP2400, UTOPIA/POS and CSIX are mutually exclusive.
- IXP2400 allows channelization; the 32 bit receive and transmit interfaces each may be divided into one, two, three, or four independent physical channels. IXP2800 does not support channelization.
- In IXP2800, RBUF may be segmented into one, two, or three segments. In IXP2400, it can be segmented into one or two, but not three, segments.
- IXP2400 defines distinct Receive Status Word formats for UTOPIA and POS modes. These formats are a subset of the one described for SPI-4 on IXP2800.
- IXP2400 defines distinct Transmit Control Word formats for UTOPIA and POS modes. These formats are a subset of the one described for SPI-4 on IXP2800.
- Due to channelization, IXP2400 has four Rx_Thread_Freelists. Each freelist is bound to a particular channel. IXP2800 has three Rx_Thread_Freelists.
- Due to channelization, IXP2400's TBUF may be partitioned into one, two, three, or four sections. Each section is bound to a particular channel. IXP2800's TBUF may be divided into one, two, or three segments.



- IXP2400's MSF_Rx_Control and MSF_Tx_Control registers have different formats.
- IXP2400 has some additional registers to configure UTOPIA/POS behavior: Rx_UP_Control_{0..3} and Tx_UP_Control_{0..3}.
- IXP2400 has additional registers to deal with MPHY, especially transmit handling: Tx_MPHY_Status, Tx_MPHY_Poll_Limit, and Rx_MPHY_Poll_Limit.
- The Rx_Cal_Length, Rx_Calendar, or Rx_Port_Status registers are specific to IXP2800. These are associated with the SPI-4 protocol.
- The Tx_Cal_Length, Tx_Calendar, or Tx_Port_Status registers only are specific to IXP2800. These are associated with the SPI-4 protocol. For transmit scheduling, in place of these registers IXP2400 has Tx_MPHY_Poll_Limit, and Tx_MPHY_Status registers, which perform a similar function.
- IXP2400 does not have Rx_Pin_Deskew registers, while IXP2800 has.
- For CSIX-L1 operation, IXP2400 only supports a fixed CWord size of 32 bits; IXP2800 supports 64, 96, and 128 bits in addition to 32 bits.

UCA Warnings

A

A.1 Introduction

This section contains the descriptions of the UCA Warnings.

Table A-1. UCA Warnings (Sheet 1 of 2)

Warning Number	Message	Section
4101	Register "REG" was defined but never used.	Appendix A.2
4700	TYPE NAME is used before being set.	Appendix A.3
4701	TYPE NAME may be used before being set.	Appendix A.4
4702	Unreachable Code.	Appendix A.5
5000	Command line option "OPTION" overrides previous option.	Appendix A.6
5002	The qualifier "any" is being ignored due to "--". Make sure Bit-16 is set in the WAKEUP_EVENTS register.	Appendix A.7
5003	Signal SIGNAL is set while already set (FILENAME:LINE).	Appendix A.8
5004	Signal SIGNAL may be set while already set (FILENAME:LINE).	Appendix A.9
5007	Doubled Signal SIGNAL is used after being consumed.	Appendix A.10
5008	TYPE NAME is set but not used.	Appendix A.11
5009	Use of TYPE transfer register REGISTER before I/O operation (FILENAME:LINE) is complete.	Appendix A.12
5011	Terminating I/O operation FILENAME:LINE at end of block because of NAME.	Appendix A.13
5012	Signal SIGNAL is doubled for ctx_arb/any. Did you really want this?	Appendix A.14
5100	REVISION is not a valid chip revision. Automatically using REVISION.	Appendix A.15
5101	Option -REVISION=REVISION overrides previous Minimum Revision of REVISION.	Appendix A.16
5102	Option -REVISION=REVISION overrides previous Maximum Revision of REVISION.	Appendix A.17
5103	Option -REVISION_MIN=REVISION overrides previous Minimum of REVISION.	Appendix A.18
5104	Option -REVISION_MAX=REVISION overrides previous Maximum of REVISION.	Appendix A.19
5114	WARNING: "CONSTANT_EXPRESSION".	Appendix A.20
5115	Manually allocated address for NAME conflicts with NAME at FILENAME:LINE	Appendix A.21
5116	Return register may not contain a valid address.	Appendix A.22
5117	Unable to determine end of operation: Queue is unknown and no signal is being generated.	Appendix A.23
5118	The use of numbered signals is obsolete and will be removed in future versions. Please use named signals: NUMBER	Appendix A.24
5121	Operand synonym "SYNONYM" hides previous ".import_var" definition.	Appendix A.25
5122	Operand synonym "SYNONYM" translated into itself.	Appendix A.26
5124	Local register "REGISTER" hides previous ".operand_synonym" definition.	Appendix A.27
5125	Local register "REGISTER" hides previous local definition.	Appendix A.28

Table A-1. UCA Warnings (Sheet 2 of 2)

Warning Number	Message	Section
5126	Local register "REGISTER" hides previous global definition.	Appendix A.29
5127	Global TYPE NAME is hidden by NAME declared at FILE LINE.	Appendix A.30
5128	Declaration for NAME hides global/module NAME declared at FILE LINE	Appendix A.31
5129	Changing "all" to "any" for ctx_arb[kill].	Appendix A.32
5130	NAME has been renamed "NAME". Future assembler versions may not support the old name.	Appendix A.33
5131	SIG_BOTH was specified, but the signal was not manually specified with .addr for %s opcode.	Appendix A.34
5132	Value of 2 specified in "TOKEN" qualifier for "cam_lookup" opcode may result in an address which exceeds the amount of Local Memory.	Appendix A.35
5133	Label LABEL is not followed by a valid uword.	Appendix A.36
5134	The directives .xfer_order_rd and .xfer_order_wr are obsolete. For sanity checking, please use ".reg read" or ".reg write".	Appendix A.37
5135	CRC type "crc_16" is not supported, defaulting to "crc_ccitt"	Appendix A.38
5136	Option -CPU= <i>n</i> will be phased out. Please use OPTION.	Appendix A.39
5137	Use of old-style Reflector Tokens will be removed in the next release. Please update your code.	Appendix A.40
5138	The "ffs" operator will be phased out for instruction "alu". Please use the "ffs" instruction instead.	Appendix A.41
5139	The "ffs" operator will be phased out for instruction "alu". Please use the "ffs" instruction instead.	Appendix A.42
5140	Reference to unreachable label "LABEL" was modified.	Appendix A.43
5141	Use of operand_synonym is obsolete and will be removed in future versions. Please use #define instead.	Appendix A.44
5142	For this chip version, writes to "REGISTER NAME" also write to the "active" version of this register.	Appendix A.45
5143	A minimum/maximum processor revision was specified when targeting multiple processor types.	Appendix A.46
5144	In the .init directive, the first data item starts with a '+'. Did you mean this to be an offset? If so, there should be no spaces before the offset.	Appendix A.47
5145	Register "NAME" should be declared as an aggregate because it is referenced as such at "FILENAME(NUMBER)".	Appendix A.48
5146	Register "NAME" should be referenced as an aggregate because it is declared as such at "FILENAME(NUMBER)".	Appendix A.49
5147	Ignoring repeated instance of specifier "SPECIFIER_NAME" for directive "DIRECTIVE_NAME".	Appendix A.50
5148	If the context for "INDIRECT_REGISTER_NAME" is swapped in, the write will not take effect because it has been placed in the defer slot of a context swapping instruction.	Appendix A.51
5149	Ignoring invalid specifier SPECIFIER for directive "DIRECTIVE".	Appendix A.52
5150	Import variable "VARIABLE" does not begin with "i\$", so "isimport()" incorrectly returned false.	Appendix A.53
5151	Declaration for "REGISTER" hides previous declaration at FILENAME(NUMBER).	Appendix A.54

A.2 UCA Warning (level 4) 4101

Register “REG” was defined but never used.

Description: The specified register was defined but was not referenced.

Example:

```
.reg x
; no other reference to x
```

How to Fix: No fix required. To get rid of the warning, delete the specified register declaration.

A.3 UCA Warning (level 1) 4700

TYPE NAME is used before being set.

Description: The specified register or signal is used before being set. The TYPE field identifies the type of NAME. The TYPE can be “Signal”, “Read Transfer Register”, “Write Transfer Register”, or “Register”.

This warning indicates that the specified register or signal is definitely being used before it is set.

Example:

```
.reg x
alu[--,--,b,x]
```

How to Fix: This warning indicates a programming error. The source code should be rewritten to set the register to an appropriate value before it is used.

A.4 UCA Warning (level 3) 4701

TYPE NAME may be used before being set.

Description: The specified register or signal may be used before being set. The TYPE field identifies the type of NAME. The TYPE can be “Signal”, “Read Transfer Register”, “Write Transfer Register”, or “Register”.

This warning indicates that the specified register or signal is set before being used on some possible paths of execution, but is used before being set on others. This may be caused by a programming error, or it may be due to the assembler assuming that all conditionals are independent.

This warning does not indicate that there is definitely a programming error, but rather that there is a possibility of a programming error.

For more details, see [Section 2.8.10, “Register Allocator Directives.”](#)

Example: Example

```
.reg r1 r2
... ; set r1
```

```
.if (r1 == 1)
immed[r2, 0]
.endif
... ; doesn't change r1
.if (r1 == 1)
alu[--,--,b,r2]
.endif
```

How to Fix: The programmer should check whether there are any valid paths where the specified register is used before being set. For example, in the above example, if the second conditional were “.if (r1 == 2)” then it would be possible for the ALU instruction to be executed without the IMMED. This indicates a programming problem, and the code should be rewritten to avoid this situation.

If, however, all valid paths do set the register before using it (in the above example, this is true because either both IFs are taken or both are skipped), then one can put a “.set NAME” immediately before the first branch, so that all paths back from the use see the set. Note that it is incorrect to put the .set between an actual assignment and the use. In this case, the assembler would assume that the actual assignment was irrelevant.

Alternately, the warning-level mechanism (e.g. a command-line argument of -W2) can be used to suppress this warning.

A.5 UCA Warning (level 2) 4702

Unreachable Code.

Description: This is the first line of a block of code that cannot be reached during normal microcode execution. This may be due to a gross programming bug, the omission of the TARGETS parameter on JUMP instructions, or by having invalid values stored in the register used in RTN.

A common situation where this can occur is within a macro that uses the .if construct, but where the macro is called with a constant rather than a register (see example below).

The presence of this warning may or may not indicate a programming error. It is considered a level-2 warning, though, because it can cause other spurious warnings to appear. In the present UCA implementation, the assembler “assumes” that execution can start at the first uword and at all of the “unreachable code” lines. This second aspect can cause spurious warnings, typically of the “used but not set” variety.

UCA handling of unreachable code will be improved in a future release.

Example:

```
#macro incr(arg, count)
.if (arg == 0)
alu[count, count, +, 1]
.else
alu[count, count, +, 2] ; unreachable
.endif
#endm
...
incr[0, count]
```

This results in the indicated line being considered unreachable. It also results in COUNT being considered used without being set at the same line.

How to Fix: If the unreachable line is due to a programming error, then the error needs to be corrected. If it is due to the situation shown in the example, then one solution would be to rewrite the macro to test ARG to see if it is a constant, and to use #if rather than .if in this case. Alternately, the warning can be ignored until handling of unreachable lines is improved. A “#pragma warning” can be used to disable this particular warning, but there is no good way to prevent spurious warnings generated by this other than using a “#pragma warning” to disable each one as it occurs.

A.6 UCA Warning (level 1) 5000

Command line option "*OPTION*" overrides previous option.

Description

Description: The command line option overrides a previous option. This warning is generated when a command line option was repeated or two mutually exclusive command line options were provided.

Example:

```
uca -ixp2400 -ixp2800 file.uc
```

How to Fix: Remove the unwanted option.

A.7 UCA Warning (level 3) 5002

The qualifier "any" is being ignored due to "--". Make sure Bit-16 is set in the WAKEUP_EVENTS register.

Description: The code contains “ctx_arb[--], any”. Because of the “--”, the “any” qualifier is being ignored. The programmer needs to make sure that in computing the value that is written into the WAKEUP_EVENTS register, that bit-16 (which indicates “any”) is being set.

In other words, when used with “ctx_arb[--]”, the “any” token has no functional significance and is merely a hint to the reader of the code. If this hint is to be accurate, then the programmer must make sure that bit-16 is set in the WAKEUP_EVENTS register.

This is not an indication of a programming error, it is a reminder to the programmer.

Example:

```
ctx_arb[--], any
```

How to Fix: Either the “any” token can be removed, or a “#pragma warning” can be used to disable this warning.

A.8 UCA Warning (level 1) 5003

Signal *SIGNAL* is set while already set (*FILENAME:LINE*).

Description: The indicated signal is definitely set while it is already set; i.e. there is a missing CTX_ARB or “BR_!SIGNAL” between the two sets. The problem here is that one does not know when the second signal arrives because it will be masked by the first signal.

This warning indicates a programming error.

Example:

```
sram[read, $x, addr,0, 1], sig_done[sig] ; This sets sig  
sram[read, $y, addr,1, 1], sig_done[sig] ; This sets sig again
```

How to Fix: A CTX_ARB, BR_SIGNAL, or BR_!SIGNAL needs to be inserted as appropriate to clear the signal between the two settings of it.

A.9 UCA Warning (level 3) 5004

Signal SIGNAL may be set while already set (FILENAME:LINE).

Description: The indicated signal is possibly set while it is already set; i.e. there is a missing CTX_ARB or “BR_!SIGNAL” between the two sets. This may be caused by a programming error, or it may be due to the assembler assuming that all conditionals are independent.

This warning does not indicate that there is definitely a programming error, but rather that there is a possibility of a programming error.

Example:

```
.set_sig s  
.if (a == 0)  
sram[read, $x, a,0, 1], sig_done[s]  
.endif  
nop  
.if (a == 0)  
ctx_arb[s]  
.endif  
;; .io_completed s  
sram[read, $x, a,0, 1], ctx_swap[s]
```

The problem occurs above because the assembler assumes that the first IF can be taken and the second one skipped, which would result in the last SRAM using the signal before it is cleared.

How to Fix: If the cause is a programming error, then that error needs to be corrected. If it is due to a situation similar to the above example, then the indicated “.io_completed” can be used to indicate that the I/O operation is completed and that the signal can be safely re-used. Alternately, after inspection, the programmer may decide to use the Warning Level or #pragma warning mechanism to mask this warning.

A.10 UCA Warning (level 1) 5007

Doubled Signal SIGNAL is used after being consumed.

Description: The indicated signal (which could be “SIGNAL+1”) is a doubled signal, and it is being used in a context where half of it was consumed on one path but not another. This is indicative of a programming error.

Example:

```
sram[swap, $x, a,0], sig_done[s] ; Signal s is doubled
.if (ctx() == 0)
l1#: br_!signal[s,l1#]          ; Consumes low half
.endif
ctx_arb[s]                    ; warning
```

The problem here is that at the `ctx_arb`, the assembler doesn't know whether it should treat `s` as being only the “high half” (i.e. `s+1`) or as “both halves”.

How to Fix: The two (or more) paths should be identified, and the code should be modified so that on all paths the signal is consumed equivalently (e.g. not consumed, partially consumed, or fully consumed).

A.11 UCA Warning (level 4) 5008

TYPE NAME is set but not used.

Description: The specified register or signal is set but not used. The TYPE field identifies the type of NAME. The TYPE can be “Signal”, “Read Transfer Register”, “Write Transfer Register”, or “Register”. A common cause would be setting the register again before it is being used or reading a transfer register and not using the results.

This warning does not necessarily indicate a programming error; it is of a more informational nature. However, it is possible that this warning is caused by a programming error.

Example:

```
immed[x, 0] ; Warning 5008
...
immed[x, 1]
...
alu[--,--,b,x]
```

The problem is that the value used in the ALU is determined by the second IMMED, and the first IMMED is therefore immaterial.

```
alu[dummy, reg, -, 5]
bne[label#]
```

The problem here is that `dummy` is not referenced and that the ALU is needed to generate a condition code.

```
.xfer_order $a $b $c
sram[read, $a, addr,0, 3], ctx_swap[sig]
alu[--,--,b,$a]
alu[--,--,b,$c]
```

The problem here is that `$b` is not being used.

How to Fix: If the assignment is not necessary but the ALU is needed for condition codes, then the destination should be “--”. Otherwise, if the assignment is truly not needed, it can be removed from the code.

In the case of the third example, the “.use” directive can be used to generate a “use” of the register without generating actual microcode.

A.12 UCA Warning (level 1) 5009

Use of *TYPE* transfer register *REGISTER* before I/O operation (*FILENAME:LINE*) is complete.

Description: A transfer register of the given type (READ/WRITE) is being used in an I/O operation, but it is being referenced before the I/O operation is known to be complete. This may work some of the time, but with different loading, this may fail.

Example:

```
immed[$x, 0]
sram[write, $x, addr,0, 1], sig_done[s]
...
immed[$x, 1] ; Warning 5009
ctx_arb[s]
```

The problem here is that if the SRAM write is suitably delayed, then the \$X can get over-written with 1 before the SRAM unit fetches the value 0.

```
sram[read, $x, addr,0, 1], sig_done[s]
ctx_arb[s], defer[1]
alu[--,--,b, $x] ; Warning 5009
```

Instructions in the defer shadow of a CTX_ARB are executed before the CTX_ARB returns, so references to the transfer registers in the defer shadow are considered to be executing before the I/O operation completes. In this case, the ALU will execute before the SRAM/read completes, and the wrong value of \$x will be used.

How to Fix: Wait until the I/O operation completes.

A.13 UCA Warning (level 1) 5011

Terminating I/O operation *FILENAME:LINE* at end of block because of *NAME*.

Description: An I/O operation was not completed when the end of a .begin/.end type of block was reached, and one of the transfer registers or signals being used in the I/O operation was scoped locally to the block.

The problem is that the registers/signals only have existence within their defining block. After the block is finished, the associated physical registers/signals are available for reuse. But if the I/O operation is not really completed, then the associated registers and signals are not really available.

Example:

```
.sig s
.begin
.reg $x
sram[read, $x, addr,0, 1], sig_done[s]
```

```

nop
nop ; Warning 5011
.end
...
ctx_arb[s]

```

The problem is that after the “.END”, \$x is still going to have a value written into it, but the programmer has indicated that use of \$x should cease outside of that block.

How to Fix: Make sure that all I/O operations are completed before the block is terminated, or declare the appropriate registers/signals within a higher block/globally.

A.14 UCA Warning (level 3) 5012

Signal *SIGNAL* is doubled for ctx_arb/any. Did you really want this?

Description: The CTX_ARB instruction was used with a doubled signal with the ANY qualifier. This is probably not what the programmer intends.

The issue is that the CTX_ARB/ANY mechanism returns when any of the specified signals is received. But since a doubled signal can come back in either order, there is no useful information that the programmer can take from this construct. That is, after this CTX_ARB returns, the programmer doesn't know which if any of the transfer registers used in the I/O operation are valid or not.

Example:

```

dram[read, $$x, addr, 0, 4], sig_done[s]
cts_arb[s], any

```

After this code completes, the programmer still has no idea of whether the I/O operation has completed or not.

How to Fix: This is probably an inappropriate use of CTX_ARB/ANY. This should probably be changed to a CTX_ARB/ALL.

A.15 UCA Warning (level 2) 5100

REVISION is not a valid chip revision. Automatically using REVISION.

Description: The chip revision provided on the command line is not valid for this release of the assembler. The assembler has chosen the nearest revision value.

Example:

```

uca --REVISION=100 file.uc

```

How to Fix: Use a valid revision number.

A.16 UCA Warning (level 2) 5101

Option `-REVISION=REVISION` overrides previous Minimum Revision of `REVISION`.

Description: The minimum revision number was already set by a previous `-REVISION` or `-REVISION_MIN` command line option. Note, the `-REVISION` option sets both the minimum and maximum target revisions to the specified value.

Example:

```
uca --REVISION_MIN=0 --REVISION=1 file.uc
```

How to Fix: Remove the unwanted revision command line option.

A.17 UCA Warning (level 2) 5102

Option `-REVISION=REVISION` overrides previous Maximum Revision of `REVISION`.

Description: The maximum revision number was already set by a previous `-REVISION` or `-REVISION_MAX` command line option. Note, the `-REVISION` option sets both the minimum and maximum target revisions to the specified value.

Example:

```
uca --REVISION=0 --REVISION=1 file.uc
```

How to Fix: Remove the unwanted revision command line option.

A.18 UCA Warning (level 2) 5103

Option `-REVISION_MIN=REVISION` overrides previous Minimum of `REVISION`.

Description: The minimum revision number was already set by a previous `-REVISION_MIN` or `-REVISION` command line option.

Example:

```
uca --REVISION=0 --REVISION_MIN=1 file.uc
```

How to Fix: Remove the unwanted revision command line option.

A.19 UCA Warning (level 2) 5104

Option `-REVISION_MAX=REVISION` overrides previous Maximum of `REVISION`.

Description: The minimum revision number was already set by a previous `-REVISION_MAX` or `-REVISION` command line option.

Example:

```
uca -REVISION_MAX=0 -REVISION_MAX=1 file.uc
```

How to Fix: Remove the unwanted revision command line option.

A.20 UCA Warning (level 1) 5114

WARNING: "CONSTANT_EXPRESSION".

Description: This warning indicates that the specified constant expression generated a warning. The only warning at present is:
Future assembler versions will not allow "=" for equality expressions. Please use "=="

Example:

```
immed[reg, (0=0)]
```

How to Fix: Replace the "=" with "==".

A.21 UCA Warning (level 1) 5115

Manually allocated address for `NAME` conflicts with `NAME` at `FILENAME:LINE`

Description: The two registers or signals named had their addresses manually allocated, they are both in use at the same time, and their addresses conflict.

The warning is reported for one of the address directives and points to the other address directive.

This warning is indicative of a programming error.

Example:

```
.reg $x $y
.addr $x 0
.addr $y 0
immed[$x,0]
immed[$y,0]
alu[--,--,b,$x]
alu[--,--,b,$y]
```

How to Fix: Assign one of the registers or signals to a different address or make sure that both registers/signals are not being used at the same time.

A.22 UCA Warning (level 1) 5116

Return register may not contain a valid address.

Description: In order for the assembler to correctly allocate registers and signals, it needs to know where RTN statements go. In order to do this, there is a requirement that the register used in the RTN instruction needs to have its value loaded with LOAD_ADDR or be a copy of such a value, and it cannot be used in a previous RTN instruction since being loaded.

For more details, see [Section 3.2.42, “RTN](#).

This warning indicates that this condition was not met. The result is that the flow-graph computed by the assembler will be incomplete, and there is a strong potential that register/signal allocation may be faulty.

This warning is indicative of a programming error.

Example:

```
load_addr[reg, lab#]
alu[reg,reg,+,1]      ; this makes the register invalid
...
rtn[reg]
```

How to Fix: Make sure that the register used in the RTN instruction always has a valid return value in it.

A.23 UCA Warning (level 2) 5117

Unable to determine end of operation: Queue is unknown and no signal is being generated.

Description: An I/O operation was issued with no signal being generated. In limited cases, this is allowed, but in general the ordering between separate I/O operations is not guaranteed, so a signal is required. Otherwise, there is no way to determine when the I/O operation completes.

Example:

```
sram[read, $x, addr, 0, 1]
```

How to Fix: Supply a signal, either through CTX_SWAP or SIG_DONE.

A.24 UCA Warning (level 2) 5118

The use of numbered signals is obsolete and will be removed in future versions. Please use named signals: NUMBER

Description: In an earlier version of the assembler, signals could be specified as numeric values. This usage has been replaced with named signals. There is no longer any reason to use numeric signals, and use of these may cause strange effects (for example, if a numeric signal is used in the context of a doubled signal).

Example:

```
sram[read, $x, addr,0, 1], sig_done[3]
```

How to Fix: Replace the signal with a named signal. In necessary, use the .ADDR directive to assign it to desired numerical value. For example:

```
.sig SIG3
.addr SIG3 3
sram[read, $x, addr,0, 1], sig_done[SIG3]
```

A.25 UCA Warning (level 3) 5121

Operand synonym "SYNONYM" hides previous ".import_var" definition.

Description: An operand_synonym directive has defined a name that matches a previous import_var name. Future references to the name will refer to the operand_synonym value rather than the imported variable.

This may or may not indicate a programming error.

Example:

```
.import_var x
.operand_synonym x y
.reg y
alu[--,--,b,x]
```

The ALU refers to the variable Y rather than the imported X.

How to Fix: For clarity, it is a good idea to not use the same name in this manner. If the programmer's intent is that future references to X should really be Y, then nothing needs to be done. Otherwise, the name of the operand_synonym should be changed.

A.26 UCA Warning (level 4) 5122

Operand synonym "SYNONYM" translated into itself.

Description: An operand synonym is equivalent either directly or indirectly to itself.

Example:

```
.operand_synonym x y
.operand_synonym y x
```

Y maps to X which maps back to Y.

How to Fix: Identify the loop and change the code to break it.

A.27 UCA Warning (level 4) 5124

Local register "REGISTER" hides previous ".operand_synonym" definition.

Description: A local register's name matches that of a previous operand_synonym. Future references will refer to the local register rather than the operand synonym.

Example:

```
.operand_synonym x y
.local x
immed[y, 0]
alu[--, --, b, x]
.endlocal
```

The ALU references local X rather than Y.

How to Fix: Use a different name.

A.28 UCA Warning (level 4) 5125

Local register "REGISTER" hides previous local definition.

Description: A register declared with .local matches the name of a similar register declared at a higher scope. Future references will be to the newly-defined register.

This warning is informative and does not indicate that a programming error exists.

Example:

```
.local x
immed[x, 0]
.local x
immed[x, 0]
.endlocal
.endlocal
```

How to Fix: Either a different name can be used, or this warning can be suppressed.

A.29 UCA Warning (level 4) 5126

Local register "REGISTER" hides previous global definition.

Description: A register declared with .local matches the name of a similar register declared at a global scope. Future references will be to the newly-defined register.

This warning is informative and does not indicate that a programming error exists.

Example:

```
.reg global x
immed[x,0]
.local x
immed[x,0]
.endlocal
```

Description: Either a different name can be used, or this warning can be suppressed.

A.30 UCA Warning (level 4) 5127

Global *TYPE NAME* is hidden by *NAME* declared at *FILE LINE*

Description: A global register is declared within the scope of a register with local scope. References to this name will refer to the local one until the scope of that local register is exited.

Example:

```
.begin
.reg x
.reg global x
immed[x,0] ; This refers to the local x, NOT the global one
.end
```

How to Fix: If the programmer wants to reference the global variable, then one of the names needs to be changed.

A.31 UCA Warning (level 4) 5128

Declaration for *NAME* hides global/module *NAME* declared at *FILE LINE*

Description: A local register is declared with the same name as a global register. References to this name will refer to the local one until the scope of that local register is exited.

This is an informational warning and does not indicate a programming problem.

Example:

```
.begin
.reg global x
.reg x
immed[x,0] ; This refers to the local x, NOT the global one
.end
```

How to Fix: If the programmer wants to reference the global variable, then one of the names needs to be changed.

A.32 UCA Warning (level 2) 5129

Changing "all" to "any" for ctx_arb[kill].

Description: The “ctx_arb[kill]” must have no optional tokens or the “any” token. If the code uses the “all” token, then the assembler changes it to “any” and generates this warning.

Example:

```
ctx_arb[kill], all
```

How to Fix: Remove the “all” token or change it to “any”.

A.33 UCA Warning (level 2) 5130

NAME has been renamed "NAME". Future assembler versions may not support the old name.

Description: During development of the assembler, some names have changed. For compatibility with earlier releases, this version still accepts the old name, but such support will be removed in future versions.

Example:

```
l#: br_inp_state[fcififo_full, l#]
```

How to Fix: Change the name as indicated.

A.34 UCA Warning (level 1) 5131

SIG_BOTH was specified, but the signal was not manually specified with .addr for %s opcode.

Description: When SIG_BOTH is specified for a reflector operation, the signal needs to be manually allocated (to the same address) in both MEs. If it is not manually allocated in the ME containing the reflect command, then this warning is generated.

This warning indicates a programming error. It is unlikely that the code will function correctly.

Example:

```
reg $x
.reg remote $r
.sig s
reflect[read, $x, 0, $r, 0, 1], sig_both, ctx_swap[s]
```

How to Fix: The signal needs to be manually allocated (using the .addr directive) to the same address in both this and the remote ME.

A.35 UCA Warning (level 2) 5132

Value of 2 specified in "TOKEN" qualifier for "cam_lookup" opcode may result in an address which exceeds the amount of Local Memory.

Description: The optional `lm_addr0` or `lm_addr1` token indicates that the lookup result should also be loaded into the given local memory address register. The immediate value provided by the token is written to bits[11:10] of the register. A `lm_addr` token value of two results in a base DWORD address of 512. Since the size of local memory is only 640 DWORDs, the result of the lookup could result in an address that exceeds the valid local memory range.

Example:

```
cam_lookup [dest, src], lm_addr0[2]
```

How to Fix: No fix is required if it is known that the lookup result will not exceed the valid memory range.

A.36 UCA Warning (level 3) 5133

Label LABEL is not followed by a valid uword.

Description: A label is applied to a directive that is not followed an actual uword/instruction. Thus, the label really applies to an undefined instruction following the assembled code.

Example:

```
nop
l#: .import_var y
; end of file
```

How to Fix: Either remove the label or add more uwords at the end of the input.

A.37 UCA Warning (level 4) 5134

The directives `.xfer_order_rd` and `.xfer_order_wr` are obsolete. For sanity checking, please use `".reg read"` or `".reg write"`.

Description: In previous versions of the assembler, a transfer register could be declared as “read-only” or “write-only” by using the `.xfer_order_rd` or `.xfer_order_wr` directives. This was supported for backwards compatibility and in order to support implicit register declarations. The preferred mechanism is to declare them with either `".reg read"` or `".reg write"`.

In the current version of the assembler, the lifetimes of transfer registers that are declared (implicitly or explicitly) as BOTH (i.e. both READ and WRITE) are tracked separately, so the only benefit of using `.xfer_order_rd` or `.xfer_ordre_wr` was checking that a read-transfer register was not written-to, and vice versa.

This feature is no longer supported. In particular, code with uses `.xfer_order_rd` or `.xfer_order_wr` to make transfer registers read or write only should still assemble in the same manner as before. The only feature lacking is that if source code changes use one of these registers in the “wrong” manner, this change would not generate an error. If this checking is desired, the code must be changed to use `".reg read"` or `".reg write"`.

Example:

```
.xfer_order_rd $x
```

How to Fix: Use “.reg read” or “.reg write”.

A.38 UCA Warning (level 1) 5135

CRC type “crc_16” is not supported, defaulting to “crc_ccitt”.

Description: In previous versions of the assembler, the CRC-CCITT polynomial was misnamed CRC-16. Specifying CRC type “crc_16” actually results in a CRC-CCITT calculation. Future versions of the assembler will generate an error when “crc_16” is specified as a CRC type.

Example:

```
crc_be [crc_16, dest, src ]
```

How to Fix: Replace the “crc_16” operand with “crc_ccitt”, for example:

```
crc_be [crc_ ccitt, dest, src ].
```

A.39 UCA Warning (level 1) 5136

Option -CPU=*n* will be phased out. Please use *OPTION*.

Description: When using the assembler from the command line, the processor type is no longer specified using the “-CPU” option. The new command line options, “-ixp2400”, “-ixp2800”, and “-ixp2xxx” should be used instead.

Example:

```
uca -CPU=2 filename.uc
```

How to Fix: Use the appropriate “-ixpDDDD” option, for example:

```
uca -ixp2800 filename.uc
```

A.40 UCA Warning (level 1) 5137

Use of old-style Reflector Tokens will be removed in the next release. Please update your code.

Description: The syntax for the sig_initiator, sig_both, and sig_remote tokens for the reflector mode of the cap instruction has changed. Please refer to section 3.2.20 for the new syntax. The old syntax will not be supported in future releases.

Example:

```
reflect[write, $xfer0, 0, $reflect_in0, 0, 1], ctx_swap [reflect_done],  
sig_initiator
```

```
reflect[write, $xfer0, 0, $reflect_in0, 0, 1], ctx_swap [reflect_done], sig_both
reflect[write, $xfer0, 0, $reflect_in0, 0, 1], sig_done [reflect_done], sig_remote
```

How to Fix: Update the syntax for the sig_initiator, sig_both, and sig_remote tokens.

A.41 UCA Warning (level 1) 5138

The "ffs" operator will be phased out for instruction "alu". Please use the "ffs" instruction instead.

Description: Previous versions of the assembler supported the ffs operator for the alu instruction in addition to the ffs instruction. Future versions of the assembler will only support the ffs instruction.

Example:

```
alu [dest, --, ffs, src]
```

How to Fix: Use the ffs instruction, for example:

```
ffs [dest, src]
```

A.42 UCA Warning (level 1) 5139

The "vnop" instruction will be phased out. Please use "nop".

Description: In previous versions of the assembler, the optimizer would remove vnop instructions but not nop instructions. Since the optimizer now removes nop instructions, there is no difference between the two, making the vnop instruction obsolete.

Example:

```
crc_be [crc_16, dest, src ]
vnop
crc_be [crc_16, dest, src ]
```

How to Fix: Replace vnop instructions with nop instructions.

A.43 UCA Warning (level 1) 5140

Reference to unreachable label "LABEL" was modified.

Description: This warning is generated when a load_addr instruction references an unreachable label. This indicates that the register loaded by the load_addr instruction was never used by a rtn instruction.

Example:

```
load_addr [rtn_addr, rtn_label#]
br [subroutine#]

rtn_label#:
```

```

alu [--, --, b, 0]
...
ctx_arb [kill]

subroutine#:
alu [--, --, b, 1]
; missing rtn [rtn_addr]

```

How to Fix: Add a rtn instruction (if missing) or remove the unnecessary load_addr instruction and the unreachable code.

A.44 UCA Warning (level 1) 5141

Use of operand_synonym is obsolete and will be removed in future versions. Please use #define instead.

Description: The operand_synonym directive has been obsolete for many releases. The operand_synonym was created to allow the illusion of an arbitrarily large register set by allowing a single physical register to be referenced by many different names. The register allocator handles this automatically. In addition, the #define preprocessor directive can be used wherever text substitution is required.

Example:

```

.operand_synonym $header $xfer0
.operand_synonym constant 0

```

How to Fix: For new registers, simply declare them with the “.reg” directive. For text substitution, simply use the #define directive. For example:

```

.reg $header
#define constant 0

```

A.45 UCA Warning (level 1) 5142

For this chip revision, writes to “REGISTER NAME” also write to the “active” version of this register.

Description: For IXP2400 and IXP2800 revisions earlier than B0, writing any of the following registers: indirect_lm_addr_0, indirect_lm_addr_0_byte_index, indirect_lm_addr_1, indirect_lm_addr_1_byte_index also writes the active version of the register: active_lm_addr_0, active_lm_addr_0_byte_index, active_lm_addr_1, active_lm_addr_1_byte_index, respectively.

Example:

```

local_csr_wr [indirect_lm_addr_0, 0]

```

How to Fix: The active version of the register should be written or rewritten after any write to the indirect register. The warning can be disabled with the #pragma warning directive. For example:

```

#if (__REVISION_MIN < __REVISION_B0)
.reg original_value
local_csr_rd [active_lm_addr_0]

```

```

immed [original_value,0]
#endif

#pragma warning (push) ; save warning settings
#pragma warning (disable: 5142)
local_csr_wr [indirect_lm_addr_0, value]
#pragma warning (pop) ; restore warning settings

#if (__REVISION_MIN < __REVISION_B0)
local_csr_wr [active_lm_addr_0, original_value]
#endif

```

A.46 UCA Warning (level 1) 5143

A minimum/maximum processor revision was specified when targeting multiple processor types.

Description: In general, specifying a minimum or maximum processor revision does not make sense when targeting multiple processor types, because processor revisions are not correlated across processor types. Note: IXP2800 revisions are correlated, but no warning is generated in this case.

Example:

```
uca -ixp2XXX -REVISION_MIN=B0 filename.uc
```

How to Fix: Remove the revision parameter or specify a specific processor type, For example:

```
uca -ixp2XXX filename.uc
uca -ixp2800 -REVISION_MIN=B0 filename.uc
```

A.47 UCA Warning (level 3) 5144

In the .init directive, the first data item starts with a '+'. Did you mean this to be an offset? If so, there should be no spaces before the offset.

Description: A '+' on a data item is interpreted as being the sign for the data item. When the assembler finds a "+" on the first data item, it is likely that a space has been inadvertently inserted between the region name and the optional offset, so the assembler reports a warning.

Example:

```
.local_mem samp_block DRAM 100
.init samp_block +1 -1
.init samp_block +16 0x1234
```

How to Fix: Either remove the space or the '+':

```
.local_mem samp_block DRAM 100
.init samp_block 1 -1
.init samp_block+16 0x1234
```

A.48 UCA Warning (level 1) 5145

Register "**NAME**" should be declared as an aggregate because it is referenced as such at "**FILENAME(NUMBER)**".

Description: Previous versions of the assembler somewhat supported aggregate notation by removing brackets from register names that were in aggregate notation, e.g. \$name[0] would become \$name0. This meant that register references could use aggregate notation, but declarations could not. Now that the assembler has true support for aggregates, the assembler will first try to match an aggregate reference to an aggregate declaration. If the register is not found, the assembler will default to the old behavior and remove the brackets. Microcode should be updated to take advantage of the aggregate support.

Example:

```
; "aggregate" declaration prior to full aggregate support
.reg $name0 $name1 $name2 $name3

immed[$name[0],0]
```

How to Fix: Replace declarations with aggregates:

```
; aggregate declaration now supported
.reg $name[4]

immed [$name[0],0]
```

A.49 UCA Warning (level 1) 5146

Register "**NAME**" should be referenced as an aggregate because it is declared as such at "**FILENAME(NUMBER)**".

Description: Previous versions of the assembler somewhat supported aggregate notation by removing brackets from register names that were in aggregate notation, e.g. \$name[0] would become \$name0. This meant that aggregate and non-aggregate names (e.g. \$name[0] and \$name0) could be used interchangeably. If a declaration is updated to use aggregate notation (perhaps in response to warning #5145), any non-aggregate references which do not match another register will match the aggregate declaration, but generate this warning. Microcode should be updated to take advantage of the full support for aggregates.

Example:

```
; declaration updated to use aggregate notation
.reg $name[4]

immed[$name[0], 0]
immed[$name1, 0] ; non-aggregate reference of register will match $name[1], but
trigger warning
```

How to Fix: Update all non-aggregate references to use aggregate notation. The example above becomes:

```
.reg $name[4]
```

```
immed[$name[0], 0]
immed[$name[1], 0]
```

A.50 UCA Warning (level 1) 5147

Ignoring repeated instance of specifier “SPECIFIER_NAME” for directive “DIRECTIVE_NAME”.

Description: A specifier for the given directive was provided more than once.

Example:

```
#pragma optimize ( "dd", off)
```

How to Fix: Remove extra instance of specifier:

```
#pragma optimize ( "d", off)
```

A.51 UCA Warning (level 2) 5148

If the context for "INDIRECT_REGISTER_NAME" is swapped in, the write will not take effect because it has been placed in the defer slot of a context swapping instruction.

Description: Due to the latency in writing local CSRs, the active CSR for the swapped in context will be loaded with the indirect CSR before the write to the indirect CSR has completed. If the swapped in context is the same as the indirect write context (CSR_CTX_POINTER), then the indirect write is effectively dropped.

Example:

```
ctx_arb[signal], defer[1]
local_csr_wr [indirect_lm_addr_0, value]
```

How to Fix: If by design, the swapped in context can never match the context selected by CSR_CTX_POINTER, then the warning can be disabled by:

```
#pragma warning (disable: 5148)
```

Otherwise, the local_csr_wr should be moved out of the defer shadow.

A.52 UCA Warning (level 2) 5149

Ignoring invalid specifier SPECIFIER for directive "DIRECTIVE".

Description: The given specifier is not one recognized by the given directive, and it is therefore being ignored.

Example:

```
#pragma optimize ("g", off)
```

How to Fix: Provide the correct specifier. If the directive is in an include file processed by another tool (e.g. a compiler) the warning can be ignored or disabled.

A.53 UCA Warning (level 1) 5150

Import variable "VARIABLE" does not begin with "i\$", so "isimport()" incorrectly returned false.

Description: The preprocessor function `isimport()` looks for an "i\$" prefix in order to determine whether a token is an import variable or not. The preprocessor must rely on this naming convention because it has no knowledge of microcode syntax and therefore does not understand `.import_var` directives. The function and naming convention were introduced with version 3.5 of the assembler.

To assist the programmer, the preprocessor maintains a list of tokens for which `isimport()` returned false. If an import variable is later declared with one of those token names, the assembler reports this warning. This warning may occur when using standard macros, as they make use of the `isimport()` function.

For more information on the new naming convention, please see the description for the `.import_var` directive.

Example:

```
#macro mymacro(a)
    #if (isimport(a))
        ; do something
    #else
        ; do something else
    #endif
#endm

.import_var noprefix

mymacro(noprefix)
```

How to Fix: Add the "i\$" prefix to the import variable:

```
.import_var i$withprefix
mymacro (i$withprefix)
```

A.54 UCA Warning (level 4) 5151

Declaration for "REGISTER" hides previous declaration at FILENAME(NUMBER).

Description: A local register is declared with the same name as another local register in an enclosing begin/end block. References to this name will refer to the one in the innermost block until the scope of that local register is exited.

This is an informational warning and does not necessarily indicate a programming problem. However, this warning may be useful in detecting the situation where a macro declares a register with the same name as a macro parameter.

Example:

```
#macro mymacro(result)
.begin
    .reg tmp
    immed[tmp, 0x1234]
    alu[result,--,b,tmp]
.end
#endm

.begin
    .reg tmp
    mymacro(tmp)
.end
```

How to Fix: If the programmer wants to reference the outermost variable, then one of the names needs to be changed. Otherwise, this warning can be ignored or disabled.

