Intel® Internet Exchange Architecture Software Building Blocks Applications

Design Guide

November 2003

Document Number: CI-00052.001

Contents

1 Introduction 19

1.1 About this Manual 19
1.2 Organization of this Manual 19
1.3 Supported Applications 20
1.4 Other Sources of Information 21

2 OC-48 POS IPv4 Forwarding Application 23

2.1 Hardware Overview 23
2.2 Software Overview 24
2.2.1 Data Flow for the Ingress IXP2400 25
2.2.1.1 Packet RX 25
2.2.1.2 PPP Decapsulation and Classify 25
2.2.1.3 IPv4 Forwarder 26
2.2.1.4 Cell Based Queue Manager (Cell QM) 26
2.2.1.5 CSIX Scheduler 27
2.2.1.6 CSIX TX 27
2.2.2 Data Flow for the Egress IXP2400 28
2.2.2.1 CSIX RX 28
2.2.2.2 PPP Encapsulation 28
2.2.2.3 Packet Based Queue Manager 28
2.2.2.4 Egress Packet WRR/DRR Scheduler 28
2.2.2.5 Packet TX 29
2.2.3 Dispatch Loops / Microblock Groups 30
2.3 Performance Characterization 31
2.4 Ingress System Resource Allocation 32
2.5 Egress System Resource Allocation 33
2.6 Interfaces Between the Various Microblocks 34
2.6.1 Packet RX and Packet Processing Microengines 35
2.6.2 Packet Processing Microengines and Cell Queue Manager 35
2.6.3 Cell Queue Manager and CSIX Scheduler 36
2.6.4 Cell Queue Manager and CSIX TX 36
2.6.5 CSIX RX and PPP Encap 37
2.6.6 PPP Encap and Packet Queue Manager 37
2.6.7 Packet Queue Manager and Scheduler 37
2.6.8 Packet Queue Manager and Packet TX 38
2.7 Core Components 38
2.7.1 Ingress Core Components 38
2.7.2 Egress Core Components 39
2.7.3 Exception Path Processing 39

3 4Gb Ethernet IPv4 Forwarding Application 41

3.1 Hardware Overview 41
3.2 Software Overview 42
3.2.1 Data Flow for the Ingress IXP2400 43
3.2.1.1 Packet RX 43
3.2.1.2 Ethernet Decapsulation/Classify/Filter 43
3.2.1.3 IPv4 Forwarder 43

3.2.1.4 Cell Based Queue Manager (Cell QM) 43
3.2.1.5 CSIX Scheduler 43
3.2.1.6 CSIX TX 43
3.2.2 Data Flow for the Egress IXP2400 43
3.2.2.1 CSIX RX 43
3.2.2.2 Ethernet Encapsulation 44
3.2.2.3 Packet Based Queue Manager (Packet QM) 44
3.2.2.4 Egress Scheduler 44
3.2.2.5 Packet TX 44
3.2.3 Dispatch Loops / Microblock Groups 44
3.2.4 Performance Characterization 46
3.3 Ingress System Resource Allocation 46
3.4 Egress System Resource Allocation 48
3.5 Interfaces Between the Various Microblocks 49
3.5.1 Packet Queue Manager and Packet TX 49
3.6 Core Components 49
3.6.1 Ingress Core Components for VxWorks 49
3.6.2 Ingress Core Components for Linux 49
3.6.3 Egress Core Components for VxWorks and Linux 49

4 OC-48 ATM IPv4 Forwarding Application 51

4.1 Hardware Overview for ATM 51
4.2 Software Overview for ATM 52
4.2.1 Data Flow for the Ingress IXP2400 52
4.2.1.1 ATM AAL5 RX 53
4.2.1.2 LLCSNAP Decapsulation and Classify 53
4.2.1.3 IPv4 Forwarder 53
4.2.1.4 Cell Based Queue Manager (Cell QM) 54
4.2.1.5 CSIX Scheduler 54
4.2.1.6 CSIX TX 54
4.2.2 Data Flow for the Egress IXP2400 54
4.2.2.1 CSIX RX 54
4.2.2.2 LLCSNAP Encapsulation 54
4.2.2.3 Cell Based Queue Manager (Cell QM) 54
4.2.2.4 Round Robin Scheduler 54
4.2.2.5 ATM AAL5 TX 56
4.2.3 Dispatch Loop 56
4.2.4 Performance Characterization 58
4.3 Ingress System Resource Allocation 58
4.4 Egress System Resource Allocation 60
4.5 Interfaces Between the Various Microblocks 61
4.5.1 AAL5 RX and Packet Processing Microengines 61
4.5.2 Packet Processing Microengines and Cell Queue Manager 61
4.5.3 Cell Queue Manager and CSIX Scheduler 61
4.5.4 Cell Queue Manager and CSIX TX 61
4.5.5 CSIX RX and LLCSNAP Encapsulation 62
4.5.6 LLCSNAP Encap and Cell Queue Manager 62
4.5.7 Cell Queue Manager and RR scheduler for ATM 63
4.5.8 RR Scheduler to Cell Queue Manager 63
4.5.9 Cell Queue Manager and AAL-5 TX 64

5 OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application 65
5.1 Hardware Overview 65
5.2 Software Overview 66
5.2.1 Data Flow for the Ingress IXP2800 67
5.2.1.1 Packet RX 67
5.2.1.2 Packet Processing Microengines (PPP Decap/Classify + IPv4/IPv6 Forwarder/Tunneling)68
5.2.1.3 Statistics Microblock 69
5.2.1.4 CSIX Scheduler 69
5.2.1.5 Cell Based Queue Manager (Cell QM) 70
5.2.1.6 CSIX TX 70
5.2.1.7 Freelist Manager 70
5.2.2 Data Flow for the Egress IXP2800 71
5.2.2.1 CSIX RX 71
5.2.2.2 Egress Packet Scheduler 71
5.2.2.3 Packet Based Queue Manager (Packet QM) 72
5.2.2.4 TX Helper 72
5.2.2.5 Packet TX 72
5.3 Performance Characterization 72
5.4 Ingress System Resource Allocation 73
5.5 Egress System Resource Allocation 74
5.6 Interfaces Between the Various Microblocks 75
5.6.1 Packet RX—First ME to Second ME 75
5.6.2 Packet RX and Packet Processing Microengines 77
5.6.3 Packet Processing Microengines and Statistics 78
5.6.4 Statistics and CSIX Scheduler 78
5.6.5 CSIX Scheduler and Cell Queue Manager 78
5.6.6 Cell Queue Manager and CSIX TX 79
5.6.7 CSIX TX—First ME to Second ME 79
5.6.8 CSIX TX (Second ME) and Freelist Manager 79
5.6.9 Freelist Manager and Packet Rx (First ME) 80
5.6.10 CSIX RX and Statistics 80
5.6.11 Statistics and Packet Scheduler 80
5.6.12 Packet Scheduler and Queue Manager 81
5.6.13 Queue Manager and TX helper 81
5.6.14 TX helper and Packet TX 81
5.6.15 Packet TX—First ME to Second ME 82
5.7 Porting from IXP2400 to IXP2800 82
5.7.1 IXP2400 and IXP2800 Processing Requirement Comparison 82
5.7.2 Optimizations for the IXP2800 83
5.7.2.1 Optimizing SRAM Memory Bandwidth Usage 83
5.7.2.2 Splitting the Packet Descriptor Across Channels 84
5.7.2.3 Splitting the RX/TX Driver Blocks to Run on Multiple Microengines 84
5.7.2.4 Moving Data Structures to Local Memory 84
5.7.2.5 Optimizing the Packet Buffer Freelist 84
5.7.2.6 Using NN Ring Instead of Scratch Ring for Communication 85
5.7.2.7 New Design for the Scheduler and Queue Manager 85
6 OC-192 POS IPv4 MPLS Application 87
6.1 Hardware Overview 87
6.2 Software Overview 88

6.2.1 Data Flow for the Ingress 89
6.2.1.1 Packet RX 89
6.2.1.2 Packet Processing Microengines (PPP Decap/Classify + MPLS ILM + IPv4 Forwarder + MPLS FTN)90
6.2.1.3 Statistics Microblock 91
6.2.1.4 CSIX Scheduler 91
6.2.1.5 Cell Based Queue Manager (Cell QM) 92
6.2.1.6 CSIX TX 92
6.2.1.7 Free List Manager 93
6.2.2 Data Flow for the Egress 93
6.3 Performance Characterization 93
6.4 Ingress System Resource Allocation 94
6.5 Egress System Resource Allocation 95
6.6 Interfaces Between the Various Microblocks 95
6.7 Application Optimizations 95
6.7.0.1 Optimizing SRAM Memory Bandwidth Usage 96
6.7.0.2 Moving Data Structures to Local Memory 96
6.7.0.3 Caching Packet Header in Local Memory 96

7 4Gb Ethernet IPv6/IPv4 Application 97

7.1 Software Overview 97
7.2 Data Flow for the Ingress IXP2400 98
7.2.1 Packet RX 98
7.2.2 Ethernet Decapsulation/Classify/Filter 98
7.2.3 V6/V4 Translation Microblock 98
7.2.4 IPv4 Forwarder 98
7.2.5 IPv6 Forwarder 99
7.2.6 IPv6/IPv4 Tunneling Microblock 99
7.2.7 Cell Based Queue Manager (Cell QM) 100
7.2.8 CSIX Scheduler 100
7.2.9 CSIX TX 100
7.3 Data Flow for the Egress IXP2400 100
7.3.1 CSIX RX 100
7.3.2 Ethernet Encapsulation 100
7.3.3 Packet Based Queue Manager (Packet QM) 100
7.3.4 Egress Scheduler 100
7.3.5 Packet TX 101
7.4 Dispatch Loops / Microblock Groups 101
7.5 Performance Analysis 102

8 DiffServ for POS Application 103

8.1 Hardware Overview 103
8.2 Software Overview 104
8.2.1 Ingress IXP2400 Network Processor - DiffServ/IPv4 104
8.2.1.1 Packet RX Microblock 105
8.2.1.2 DiffServ/IPv4 Functional Pipeline 106
8.2.1.3 Ingress Queue Manager for DiffServ 107
8.2.1.4 CSIX Scheduler 107
8.2.1.5 CSIX TX Microblock 107
8.2.2 Egress IXP2400 Network Processor—DiffServ/ IPv4 108
8.2.2.1 CSIX RX Microblock 108
8.2.2.2 DiffServ Functional Pipeline 108

8.2.2.3 Egress Queue Manager 109
8.2.2.4 Egress Scheduler 109
8.2.2.5 Packet TX Microblock 109
8.2.3 Performance Analysis 109
8.3 System Data Structures and Resource Allocation 109
8.3.1 Ingress System Resource Allocation 110
8.3.2 Egress System Resource Allocation 110
8.3.3 Buffer Handle 111
8.3.4 Packet Metadata 111
8.4 Interfaces Between the Various Microblocks 112
8.4.1 Inter-Microengine Messages 112
8.4.1.1 POS RX and Ingress DiffServ/IPv4 Functional Pipeline 112
8.4.1.2 Ingress DiffServ/IPv4 Functional Pipeline and Ingress Queue Manager 112
8.4.1.3 Ingress Queue Manager and Ingress Scheduler 112
8.4.1.4 Ingress Queue Manager and CSIX TX 113
8.4.1.5 CSIX RX and Egress DiffServ Pipeline 113
8.4.1.6 Egress DiffServ Pipeline and Egress Queue Manager 113
8.4.1.7 Egress Queue Manager and Scheduler 113
8.4.1.8 Egress Queue Manager and POS TX 114
8.4.2 Ingress Dispatch Loop Variables 114
8.4.3 Egress Dispatch Loop Variables 115
8.5 Dynamic Behavior 116
8.5.1 Ingress Data Flow 116
8.5.1.1 Ingress Core Components 118
8.5.2 Egress Data Flow 120
8.5.2.1 Microblock Egress Pipeline 120
8.5.2.2 Egress Core Components 121
8.6 Sending Packets from Core Components to Microblocks 122
8.7 Statistics Handling 124

9 DiffServ for ATM Application 127

9.1 Hardware Architecture 127
9.2 Software Architecture 127
9.2.1 Ingress IXP2400 127
9.2.1.1 Ingress Microblock Pipeline 128
9.2.1.2 Ingress Core Components 130
9.2.2 Egress IXP2400 130
9.2.3 Performance Analysis 131
9.3 System Data Structures, Interfaces, and Resource Allocation 131
9.3.1 Ingress System Resource Allocation 132
9.3.2 Egress System Resource Allocation 132
9.3.3 Buffer Handle 133
9.3.4 Packet Metadata 133
9.4 Interfaces Between the Various Microblocks 133
9.4.1 Inter-Microengine Messages 133
9.4.1.1 AAL5 RX and Ingress DiffServ/IPv4 Functional Pipeline 133
9.4.1.2 Ingress DiffServ/IPv4 Functional Pipeline and Ingress Queue Manager 133
9.4.1.3 Ingress Queue Manager and Ingress Scheduler 134
9.4.1.4 Ingress Queue Manager and CSIX TX 134
9.4.1.5 CSIX RX and Egress DiffServ Pipeline 134
9.4.1.6 Egress DiffServ pipeline and Egress Cell Queue Manager 134
9.4.1.7 Egress Cell Queue Manager and TM4.1 Shaper 134

9.4.1.8 TM4.1 Shaper and TM 4.1 Writeout/Scheduler 134
9.4.1.9 RR Scheduler and Egress Cell Queue Manager 134
9.4.1.10 Egress Queue Manager and AAL5 TX 134
9.4.2 Ingress Dispatch Loop Variables 134
9.4.3 Egress Dispatch Loop Variables 134

10 MPLS Application 135

10.1 Input/Output Media Independence 135
10.2 MPLS Forwarder Decomposition 136
10.2.1 Ingress LER Generic MPLS Forwarder 136
10.2.2 LSR Generic MPLS Forwarder 137
10.2.3 Egress LER Generic MPLS Forwarder 138
10.2.4 MPLS Forwarder Building Blocks 138
10.3 Cooperation with IP and QoS Microblocks 139
10.3.1 IP and MPLS Functional Pipeline 140
10.3.2 TTL Processing 141
10.3.2.1 TTL Processing in Different Tunneling Models 142
10.4 Data Plane Architecture Dependencies 144
10.4.1 Target HW Architecture 144
10.4.2 Ingress and Egress Microblocks 145
10.4.3 MPLS Forwarder Core Component Overview 147
10.4.3.1 Inter-Component Dependencies 148

11 10 Gb Ethernet IPv4/IPv6 Forwarding/Tunneling Application 151

11.1 Hardware Overview 151
11.2 Software Overview 153
11.2.1 Data Flow for the Ingress IXP2800 153
11.2.1.1 Packet RX 153
11.2.1.2 Packet Processing Microengines (PPP Decap/Classify + IPv4/IPv6 Forwarding + Tunneling)154
11.2.1.3 Statistics Microblock 154
11.2.1.4 CSIX Scheduler 154
11.2.1.5 Cell Based Queue Manager (Cell QM) 154
11.2.1.6 CSIX TX 154
11.2.2 Data Flow for the Egress IXP2800 154
11.2.2.1 CSIX RX 154
11.2.2.2 Ethernet ARP Microblock 155
11.2.2.3 Statistics Microblock 155
11.2.2.4 Egress Packet Scheduler 155
11.2.2.5 Packet Based Queue Manager (Packet QM) 155
11.2.2.6 TX Helper 155
11.2.2.7 Packet TX 155
11.3 Performance Characterization 156
11.4 Ingress System Resource Allocation 157
11.5 Egress System Resource Allocation 158
11.6 Interfaces Between the Various Microblocks 159
11.6.1 Packet RX and Packet Processing Microengines 160
11.6.2 Packet Processing Microengines and Statistics 160
11.6.3 Statistics and CSIX Scheduler 161
11.6.4 CSIX Scheduler and Cell Queue Manager 161
11.6.5 Cell Queue Manager and CSIX TX 161
11.6.6 CSIX TX—First ME to Second ME 162

11.6.7 CSIX RX and Ethernet ARP 162
11.6.8 Ethernet ARP and Statistics 163
11.6.9 Statistics and Packet Scheduler 163
11.6.10 Packet Scheduler and Queue Manager 163
11.6.11 Queue Manager and TX helper 164
11.6.12 TX Helper and Packet TX (10x1 GigE) 164
11.6.13 TX Helper and Packet TX (1x10 GigE) 164
11.6.14 Packet TX —First ME to Second ME (1x10 GigE) 165

12 Core Router Application 167

12.1 Hardware Overview 167
12.2 Software Overview 168
12.2.1 Data Flow for the Ingress A IXP2800 169
12.2.1.1 Packet RX 169
12.2.1.2 Packet Processing Microengines 169
12.2.1.3 Dispatch Loop / Microblock Groups 173
12.2.1.4 Statistics 173
12.2.1.5 SPI4 TX 173
12.2.2 Data Flow for the Ingress B IXP2800 174
12.2.2.1 SPI4 RX 174
12.2.2.2 Meter & WRED 174
12.2.2.3 Statistics Microblock 174
12.2.2.4 CSIX Scheduler 174
12.2.2.5 Cell Based Queue Manager (Cell QM) 174
12.2.2.6 CSIX TX 174
12.2.3 Data Flow for the Egress IXP2800 174
12.2.3.1 CSIX RX 175
12.2.3.2 Statistics Microblock 175
12.2.3.3 Egress Packet Scheduler 175
12.2.3.4 Packet Based Queue Manager (Packet QM) 175
12.2.3.5 TX Helper 175
12.2.3.6 Packet TX 176
12.3 Performance Characterization 176
12.4 Ingress A System Resource Allocation 176
12.5 Ingress B System Resource Allocation 177
12.6 Egress System Resource Allocation 178
12.7 Interfaces Between the Various Microblocks 179
12.7.1 Packet RX and Packet Processing Microengines 179
12.7.2 Packet Processing Microengines and Statistics 180
12.7.3 Statistics and SPI4 TX 180
12.7.4 SPI4 RX and Meter/WRED 181
12.7.5 METER/WRED and Statistics 181
12.7.6 Statistics and CSIX Scheduler 182
12.7.7 CSIX Scheduler and Cell Queue Manager 182
12.7.8 Cell Queue Manager and CSIX TX 182
12.7.9 CSIX TX—First ME to Second ME 183
12.7.10 CSIX RX and Statistics 183
12.7.11 Statistics and Packet Scheduler 183
12.7.12 Packet Scheduler and Queue Manager 184
12.7.13 Queue Manager and TX Helper 184
12.7.14 TX Helper and Packet TX 184

13 Dual OC-12 POS/Dual Gb Ethernet Forwarding Application for IXDP24X1 187
13.1 Hardware Overview 187
13.2 Software Overview 189
13.2.1 Data Flow 189
13.2.1.1 Ethernet Packet RX 189
13.2.1.2 POS RX 190
13.2.1.3 Ethernet Decapsulation and Classify 190
13.2.1.4 PPP Decapsulation and Classify 191
13.2.1.5 IPv4 Forwarder 191
13.2.1.6 Packet-Based Queue Manager 191
13.2.1.7 Packet Scheduler 192
13.2.1.8 Ethernet Encapsulation 192
13.2.1.9 Ethernet Packet TX 192
13.2.1.10 PPP Encapsulation 193
13.2.1.11 POS Packet TX 193
13.2.2 Dispatch Loops 193
13.3 Performance Characterization 195
13.3.1 POS/Ethernet Pipeline 195
13.4 System Resource Allocation 196
13.5 Microblock Interface 197
13.5.1 Packet RX and Packet Processing Microengines 197
13.5.2 Packet Processing Microengines and Packet QM 198
13.5.3 Packet Queue Manager and Scheduler 198
13.5.4 Packet Queue Manager and POS Packet TX 199
13.5.5 Packet Queue Manager and Ethernet Packet TX 199
13.6 Core Components Usage 199
14 Quad Gigabit Ethernet Forwarding Application for IXDP24X1 201
14.1 Hardware Overview 201
14.2 Software Overview 203
14.2.1 Data Flow 203
14.2.1.1 Packet RX 203
14.2.1.2 Ethernet Classify/Decapsulate 204
14.2.1.3 IPv4 Forwarder 204
14.2.1.4 L2 Validate 204
14.2.1.5 Packet-Based Queue Manager 204
14.2.1.6 Packet Scheduler 204
14.2.1.7 Ethernet Encapsulation 205
14.2.1.8 Packet TX 205
14.2.2 Dispatch Loops 205
14.2.3 HW Architecture-Specific Code 206
14.2.3.1 Quad Gigabit Ethernet MAC Driver 206
14.2.3.2 Ethernet PHY Driver 207
14.3 Performance Characterization 208
14.4 System Resource Allocation 208
14.5 Microblock Interfaces 209
14.5.1 Packet RX and Packet Processing Microengines 210
14.5.2 Packet Processing Microengines and Packet QM 210
14.5.3 Packet Scheduler and Packet QM 210
14.5.4 Packet Queue Manager and Packet TX 210
14.6 Core Component Usage 210

15 ATM/Ethernet IPv4 Forwarding Application for IXDP24X1 213
15.1 Hardware Overview 213
15.2 Software Overview 214
15.3 Data Flow 215
15.3.1 AAL5 RX/Ethernet RX 215
15.3.2 Packet Processing 216
15.3.3 Packet-Based Queue Manager 216
15.3.4 Packet Scheduler 216
15.3.5 Cell-Based Queue Manager 216
15.3.6 TM 4.1 Shaper 217
15.3.7 TM 4.1 Cell Scheduler 217
15.3.8 Ethernet Tx 217
15.3.9 ATM AAL5 Tx 217
15.4 Dispatch Loops 218
15.5 Performance Characterization 220
15.6 System Resource Allocation 221
15.7 Microblock Interfaces 222
15.7.1 Common RX to Packet Processing 222
15.7.2 Packet Processing to Packet Queue Manager 223
15.7.3 Scheduler to Queue Manager 223
15.7.4 Queue Manager to Scheduler 223
15.7.5 Queue Manager to Packet TX 223
15.7.6 Queue Manager to TM 4.1 Shaper 224
15.7.7 TM4.1 Scheduler to Queue Manager 224
15.7.8 Queue Manager to TM4.1 Scheduler 225
15.7.9 Queue Manager to AAL5 TX 225

16 POS/Ethernet IPv4 Forwarding Application for IXDP28x1 227
16.1 Hardware Overview 227
16.2 Software Overview 229
16.2.1 Data Flow 230
16.2.1.1 Packet RX 230
16.2.1.2 Packet Processing 230
16.2.1.3 Packet-Based Queue Manager 231
16.2.1.4 Packet Scheduler 231
16.2.1.5 Packet TX 231
16.2.2 `Dispatch` Loops 232
16.3 Performance Characterization 233
16.4 System Resource Allocation 233
16.5 Microblock Interfaces 235
16.5.1 Packet RX to Packet Processing Microengine 235
16.5.2 Packet Processing to Queue Manager Microengine 236
16.5.3 Scheduler to Queue Manager Microengine 236
16.5.4 Queue Manager to Scheduler Microengine 236
16.5.5 Queue Manager to Packet TX Microengine 237
16.6 Core Components Integration 237

Figures

2-1 Example Hardware Configuration for OC48-POS with CSIX Fabric 23

2-2 Microblocks for an OC-48 POS IPv4 Forwarding Application 24
2-3 Dispatch Loop for the Packet Frame Reassembly Stage 30
2-4 Dispatch Loop for the IPv4 Functional Pipeline 30
2-5 Dispatch Loop for CSIX Reassembly Stage 31
2-6 Dispatch Loop for POS Transmit Stage 31
3-1 Example Hardware Configuration for 4x1 Gigabit Ethernet with CSIX Fabric 41
3-2 Software Components for IPv4 Forwarding Application for Ethernet 42
3-3 Dispatch Loop for the Packet Frame Reassembly Stage 44
3-4 Dispatch Loop for the IPv4 Functional Pipeline 45
3-5 Dispatch Loop for CSIX Reassembly Stage 45
3-6 Dispatch Loop for Ethernet Encapsulation Stage 45
4-1 Example Hardware Configuration for OC-48 ATM with CSIX Fabric 51
4-2 Software Components for IPv4 Forwarding Application for OC-48 ATM 52
4-3 Dispatch Loop for the AAL5 Reassembly Stage 56
4-4 Dispatch Loop for the IPv4 Functional Pipeline 57
4-5 Dispatch Loop for CSIX Reassembly Stage 57
4-6 Dispatch Loop for LLCSNAP Encapsulation Stage 57
5-1 Example Hardware Configuration for OC-192 POS Line Card with CSIX Fabric 66
5-2 Microblocks for an OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application 67
6-1 Example Hardware Configuration for OC-192 POS Line Card with CSIX Fabric 88
6-2 Microblocks for an OC-192 POS IPv4 MPLS Forwarding Application 89
7-1 Software Components for IPv4/IPv6 Forwarding and IPv6/IPv4 Tunneling 97
7-2 Dispatch Loop for the Packet Frame Reassembly Stage 101
7-3 Dispatch Loop for the IPv4, IPv6 and V6/V4 Tunneling Functional Pipeline 101
7-4 Dispatch Loop for CSIX Reassembly Stage 102
7-5 Dispatch Loop for Ethernet Encapsulation Stage 102
8-1 IXDP2400 Advanced Development Platform Overview 104
8-2 IPv4 and DiffServ—Ingress Blocks 105
8-3 IPv4 and DiffServ: Egress Architecture 108
8-4 Slow path for POS DiffServ Application 119
8-5 Handling Xscale Scratch Ring by the Source Macro 123
8-6 Statistics Update—Atomic Operations and Within a Critical Section 124
9-1 Software Architecture of ATM/DiffServ/IPv4 blocks on the Ingress Processor 128
9-2 ATM, IPv4 and DiffServ—Egress Architecture 130
10-1 MPLS Flow Processing 135
10-2 MPLS Forwarding: Data Path for Ingress LER 136
10-3 MPLS Forwarding: Data Path for LSR 137
10-4 MPLS Forwarding: Data Path for Egress LER 138
10-5 MPLS Forwarder Building Block 139
10-6 Universal IP and MPLS Microblocks Layout 140
10-7 TTL Processing for Uniform Model LSPs 142
10-8 TTL Processing for Short Pipe Model LSPs without PHP 142
10-9 TTL Processing for Short Pipe Model LSPs with PHP 143
10-10 TTL Processing for Pipe Model LSPs without PHP 143
10-11 Dual NP Blade Architecture 144
10-12 Single NP Blade Architecture 145
10-13 MPLS Ingress and Egress Microblocks 146
10-14 MPLS Forwarder Core Component 147
10-15 MPLS Core Component Initialization Order 149
10-16 MPLS Core Component De-Initialization Order 149

11-1 Example Hardware Configuration for 10x1/1x10 Gb Ethernet with CSIX Fabric 152
11-2 Microblocks for an Ethernet 10x1/1x10 Gb IPv4 Forwarding Application 153
12-1 Example Hardware Configuration for Core Metro Application Using 3 IXP2800 168
12-2 Microblocks for Core Router Application 169
12-3 Packet Flow in Ingress A 172
13-1 Example Hardware Configuration for OC48-Ethernet/POS 188
13-2 Microblocks for Dual OC-12 POS/ Dual Gigabit Ethernet IPv4 Forwarding Application 189
13-3 Dispatch Loop for the Packet Frame Reassembly Stage 194
13-4 Dispatch Loop for the IPv4 Functional Pipeline 194
13-5 Dispatch Loop for POS Transmit Stage 194
13-6 Dispatch Loop (Microblock group) for Ethernet Transmit Stage 195
13-7 Core Components in the OC-12 POS/Ethernet IPv4 Forwarding Application 200
14-1 Example Hardware Configuration for Quad Ethernet IPv4 Forwarding Application 202
14-2 Microblocks for a Quad Ethernet IPv4 Forwarding Application 203
14-3 Dispatch Loop for the Packet Frame Reassembly Stage 206
14-4 Dispatch Loop for the IPv4 Functional Pipeline 206
14-5 Dispatch Loop for the Ethernet Transmit Stage 206
14-6 Ethernet Interface Connections to Quad Gigabit Ethernet MAC Device 207
14-7 Core Components in the Quad Ethernet IPv4 Forwarding Application 211
15-1 ATM-IPv4-Ethernet Application Microblocks 214
15-2 Dispatch Loop for Ethernet Receive Stage 218
15-3 Dispatch Loop for ATM Receive Stage 218
15-4 Dispatch Loop for IPv4 Forwarder Packet Processing (Ethernet to ATM) 219
15-5 Dispatch Loop for IPv4 Forwarder Packet Processing (ATM to Ethernet) 219
15-6 Dispatch Loop for ATM Transmit Stage 220
15-7 Dispatch Loop for Ethernet Transmit Stage 220
16-1 POS/Ethernet Hardware Configuration on IXDP28X1 229
16-2 POS-Ethernet IPv4 Application Microblocks 230
16-3 Dispatch Loop for the Packet Frame Reassembly Stage 232
16-4 Dispatch Loop for IPv4 Forwarder Packet Processing 232
16-5 POS-Ethernet IPv4 Application Core Components 237

Tables

1-1 Supported Applications 20
2-1 Performance Characterization for the POS Pipeline 31
2-2 System Resources Mapped for the Ingress IXP2400 32
2-3 SRAM, DRAM and Scratch Utilization for Ingress System Resources 32
2-4 System Resources Allocated for the Egress IXP2400 33
2-5 SRAM, DRAM and Scratch Utilization for Egress System Resources 34
2-6 Five-Word Entry in Scratch Ring (IPv4 Forwarder + PPP Decap) 35
2-7 Three-Word Entry in Scratch Ring (IPv4 Forwarder + PPP Decap) 35
2-8 Dequeue Requests via the Scratch Ring 36
2-9 Queue Transition Messages Sent by the Queue Manager 36
2-10 Two-Word Entry in Scratch Ring 36
2-11 Three-Word Entry in Scratch Ring (CSIX and PPP Encap) 37
2-12 Scratch Ring Interface between PPP Encap and Packet Queue Manager 37
2-13 Queue Transition Messages Sent by the Packet Queue Manager 37
2-14 One-word Scratch Ring Entry 38
3-1 Performance Characterization for the Ethernet Pipeline 46

3-2 Ingress System Resources Mapped for the Ingress IXP2400 46
3-3 SRAM, DRAM, and Scratch Utilization for Ingress System Resource Allocation 47
3-4 System Resources Allocated for the Egress IXP2400 48
3-5 SRAM, DRAM, and Scratch Utilized for Egress System 48
3-6 One-Word Scratch Ring (Packet Queue Manager and Packet TX) 49
4-1 Performance Characterization for the ATM Pipeline 58
4-2 System Resources Mapped for the Ingress IXP2400 58
4-3 SRAM and DRAM Utilization for Ingress System Resource Allocation 59
4-4 System Resources Allocated for the Egress IXP2400 60
4-5 SRAM and DRAM Utilization for Egress System Resource Allocation 60
4-6 Six-Word Scratch Ring Entry (IPv4+L2 Decap) 61
4-7 Three-Word Scratch Ring (CSIX RX and LLCSNAP Encap) 62
4-8 Three-Word Scratch Ring (LLCSNAP Encap and Cell QM) 62
4-9 Cell Queue Manager and RR scheduler for ATM 63
4-10 One-Word Scratch Ring Entry (TM 4.1 Scheduler to Cell Queue Manager) 63
4-11 Two-Word Scratch Ring Entry (Cell Queue Manager and AAL-5 TX) 64
5-1 Performance Analysis for the POS Pipeline 72
5-2 System Resources Mapped for the Ingress IXP2800 73
5-3 SRAM, DRAM, and Scratch Utilization for Ingress IXP2800 73
5-4 System Resources Allocated for Egress IXP2800 74
5-5 SRAM, DRAM, and Scratch Utilization for Egress IXP2800 75
5-6 Five-Word NN Ring Entry (Packet RX—First ME to Second ME) 76
5-7 Three-Word Scratch Ring Entry —Packets fit on one Buffer 77
5-8 Three-Word Scratch Ring Entry —Packets Require more than one Buffer 77
5-9 Three-Word Scratch Ring Entry—Packet Processing Microengines and Statistics 78
5-10 Three-Word NN Ring Entry (Statistics and CSIX Scheduler) 78
5-11 Three-Word NN Ring Entry (CSIX Scheduler and Cell Queue Manager) 78
5-12 Two-Word NN Ring Entry (Cell Queue Manager and CSIX TX) 79
5-13 Eight-Word NN Ring Entry (CSIX TX—First ME to Second ME) 79
5-14 One-Word NN Ring Entry 79
5-15 One-word NN Ring Entry 80
5-16 Three-Word Scratch Ring Entry (CSIX RX and Statistics) 80
5-17 Three-Word NN Ring Entry (Statistics and Packet Scheduler) 80
5-18 Three-word NN Ring Entry (Queue Manager and Packet Scheduler) 81
5-19 Two-Word NN Ring Entry (Queue Manager and Packet TX) 81
5-20 One-Word NN Ring Entry (Queue Manager and Packet TX) 81
5-21 Three-Word NN Ring Entry (Packet TX—First ME to Second ME) 82
5-22 Three-Word NN Ring Entry (for Non-stop m-packet) 82
5-23 Comparison of IXP2400 and the IPX2800 Processing Requirements 82
5-24 Data Structure Location Comparison of the IXP2400 and IXP2800 Applications 83
6-1 Performance Analysis for the POS Pipeline 93
6-2 System Resources Mapped for the Ingress IXP2800 94
6-3 SRAM, DRAM, and Scratch Utilization for Ingress IXP2800 94
6-4 Data Structure Allocations 96
7-1 Performance Analysis for the IPv6 Ethernet Pipeline 102
8-1 Ingress IXP2400 Memory Usage 110
8-2 Egress IXP2400 Microengine Allocation 110
8-3 Egress IXP2400 Memory Usage 111
8-4 Packet Metadata Structure 111
8-5 Message Format Between CSIX RX and Egress DiffServ Pipeline 113

8-6 Message Format Between Egress Queue Manager and Scheduler 113
8-7 Ingress Dispatch Loop Variables for DiffServ Application 114
8-8 Egress Dispatch Loop Variables for DiffServ Application 115
8-9 IPv4 and Diffserv Ingress Dispatch Loop Variables 116
8-10 MPLS, IPv4 and Diffserv Egress Dispatch Loop Variables 121
8-11 DiffServ Core Components to Diffserv Pipeline Message Fields 122
8-12 Statistics Overhead at SRAM Controller 125
9-1 Ingress IXP2400 Memory Usage 132
9-2 Egress IXP2400 Microengine Allocation 132
9-3 Egress IXP2400 Memory Usage 133
11-1 Summary of Performance Analysis for the Ethernet Pipeline 156
11-2 System Resources Mapped for the Ingress IXP2800 157
11-3 SRAM, DRAM and Scratch Utilized for Ingress IXP2800 157
11-4 System Resources Allocation for the Egress IXP2800 158
11-5 SRAM, DRAM and Scratch Utilization for Egress IXP2800 159
11-6 Three-Word Scratch Ring Entry —Packets fit on one Buffer 160
11-7 Three-Word Scratch Ring Entry —Packets Require more than one Buffer 160
11-8 Three-Word Scratch Ring Entry—Packet Processing Microengines and Statistics 160
11-9 Three-Word NN Ring Entry (Statistics and CSIX Scheduler) 161
11-10 Three-Word NN Ring Entry (CSIX Scheduler and Cell Queue Manager) 161
11-11 Two-Word NN Ring Entry (Cell Queue Manager and CSIX TX) 161
11-12 Eight-Word NN Ring Entry (CSIX TX—First ME to Second ME) 162
11-13 Three-Word Scratch Ring Entry (CSIX RX and Statistics) 162
11-14 Three-Word Scratch Ring Entry (Statistics and Ethernet ARP) 163
11-15 Three-Word NN Ring Entry (Statistics and Packet Scheduler) 163
11-16 Three-word NN Ring Entry (Queue Manager and Packet Scheduler) 163
11-17 Two-Word NN Ring Entry (Queue Manager and Packet TX) 164
11-18 Two Scratch Ring Interface (TX Helper and Packet TX)—One Word 164
11-19 One-Word NN Ring Entry (Queue Manager and Packet TX) 164
11-20 Three-Word NN Ring Entry (Packet TX—First ME to Second ME) 165
11-21 Three-Word NN Ring Entry (for Non-stop m-packet) 165
12-1 Performance Analysis for the POS Pipeline 176
12-2 System Resources Mapped for the Ingress IXP2800 176
12-3 SRAM, DRAM, and Scratch Utilization for Ingress A IXP2800 177
12-4 System Resources Allocated for Ingress B IXP2800 177
12-5 SRAM, DRAM, and Scratch Utilization for Ingress B IXP2800 178
12-6 System Resources Allocated for Egress IXP2800 178
12-7 SRAM, DRAM, and Scratch Utilization for Egress IXP2800 179
12-8 Three-Word Scratch Ring Entry—Packets fit on one Buffer 179
12-9 Three-Word Scratch Ring Entry—Packets fit on more than one Buffer 180
12-10 Three-Word Scratch Ring Entry—Packet Processing Microengines and Statistics 180
12-11 Three-Word NN Ring Entry (Statistics and SPI4 TX) 180
12-12 Three-Word Scratch Ring Entry (One Buffer only) 181
12-13 Three-Word Scratch Ring Entry for SPI4 RX and Meter/WRED 181
12-14 Three-Word Scratch Ring Entry for Meter/WRED and Statistics 181
12-15 Three-Word NN Ring Entry for Statistics and CSIX Scheduler 182
12-16 Three-Word NN Ring Entry for CSIX Scheduler and Cell Queue Manager 182
12-17 Two-Word NN Ring Entry for Cell Queue Manager and CSIX TX 182
12-18 Eight-Word NN Ring Entry (CSIX TX—First ME to Second ME) 183
12-19 Three-Word Scratch Ring Entry for CSIX RX and Statistics 183

12-20 Three-Word NN Ring Entry for Statistics and Packet Scheduler 183
12-21 Two-Word NN Ring Entry for Packet Scheduler and Queue Manager 184
12-22 Two-Word NN Ring Entry for Queue Manager and TX Helper 184
12-23 One-Word Scratch Ring Entry for TX Helper and Packet TX 184
13-1 Supported Hardware Configurations 188
13-2 Performance Characterization for the POS Pipeline 195
13-3 System Resources Mapped for the IXP2400 196
13-4 SRAM, DRAM, and Scratch Utilization for Ingress Resource Allocation 196
13-5 Packet RX and Packet Processing Microengines Scratch Ring Interface 197
13-6 Packet Processing Microengines and Packet QM Scratch Ring Interface 198
13-7 Queue Transition Messages Sent by the Packet Queue Manager 198
13-8 Packet Queue Manager and Packet TxScratch Ring Interface 199
13-9 One-word Scratch Ring (Packet Queue Manager and Packet TX) 199
14-1 Supported Hardware Configurations 202
14-2 Performance Characterization for the Ethernet Pipeline 208
14-3 System Resources Mapped for the IXP2400 208
14-4 SRAM, DRAM and Scratch Utilization for System Resource Allocation 209
14-5 One-word Scratch Ring (Packet Queue Manager and Packet TX) 210
15-1 Supported Hardware Configurations 213
15-2 System Resources Mapped for the Intel® IXP2400 Network Processor 221
15-3 SRAM Memory Map 221
15-4 Common RX to Packet Processing Microengines Scratch Ring Interface 222
15-5 Packet Processing to Packet Queue Manager Scratch Ring Interface 223
15-6 Scheduler to Queue Manager Scratch Ring Interface 223
15-7 Queue Manager to Scheduler Next Neighbor Ring Interface 223
15-8 Queue Manager to Packet TX Scratch Ring Interface 223
15-9 Queue Manager to TM 4.1 Shaper Next Neighbor Ring Interface 224
15-10 TM4.1 Scheduler to Queue Manager Scratch Ring Interface 224
15-11 Queue Manager to TM4.1 Scheduler Scratch Ring Interface 225
15-12 Queue Manager to AAL5 TX Scratch Ring Interface 225
16-1 Supported Hardware Configurations 228
16-2 Performance Characterization for the POS-Ethernet IPv4 Application 233
16-3 System Resources Mapped for the Intel® IXP2800 Network Processor 233
16-4 SRAM Memory Map 234
16-5 SRAM Channels Budget For Packets Processing with Minimal Length† 234
16-6 Packet RX to Packet Processing Microengine Scratch Ring Interface 235
16-7 Packet Processing to Queue Manager Microengine Scratch Ring Interface 236
16-8 Scheduler to Queue Manager Microengine Scratch Ring Interface 236
16-9 Queue Manager to Scheduler Microengine Next Neighbor Ring Interface 236
16-10 Queue Manager to Packet TX Microengine Scratch Ring Interface 237

Revision History

Date	Revision	Description
July 2003	Initial release	Included with IXA SDK 3.1 Field Trial Release. Application chapters were removed from the *Intel® Internet Exchange Architecture Software Building Blocks Developer's Manual* to create this new book. New applications in this release include: DiffServ for ATM, 10Gb Ethernet IPv4 Forwarding, Core Router, Dual OC-12 POS/ Dual Gigabit Ethernet Forwarding, and Quad Gigabit Forwarding.
November 2003	IXA SDK 3.5	New applications in this release: • OC-192 POS IPv4 MPLS Application • ATM/Ethernet IPv4 Forwarding Application for Intel® IXDP24x1 • POS/Ethernet IPv4 Forwarding Application for Intel® IXDP28x1 Changes in this release: • OC-192 POS IPv4 Forwarding Application has new IPv4/IPv6 Forwarding and Tunneling functionality. • 10 Gb Ethernet IPv4 Forwarding application has new IPv4/IPv6 Forwarding and Tunneling functionality.

Introduction 1

1.1 About this Manual

The Intel® Internet Exchange Architecture Software Development Kit (Intel® IXA SDK) includes example code that demonstrates network processor features and data flow. These applications can be used to jump-start customer application development.

This manual provides a high-level design overview of the hardware and software components for applications developed specifically for the Intel® IXP2400 Network Processor and the Intel® IXP2800 Network Processor. This manual focuses on the microengine components of each design, listing the building blocks used, describing the data flow of the application, and providing performance characterization data.

1.2 Organization of this Manual

Chapter 1, "Introduction," (this chapter) describes how this manual is organized and lists other manuals which may be referred to for more information.

The remaining chapters in this manual describe the design details, data flow descriptions, and performance characterization of the following software applications:

- Chapter 2, "OC-48 POS IPv4 Forwarding Application"
- Chapter 3, "4Gb Ethernet IPv4 Forwarding Application"
- Chapter 4, "OC-48 ATM IPv4 Forwarding Application"
- Chapter 5, "OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application"
- Chapter 6, "OC-192 POS IPv4 MPLS Application"
- Chapter 7, "4Gb Ethernet IPv6/IPv4 Application"
- Chapter 8, "DiffServ for POS Application"
- Chapter 9, "DiffServ for ATM Application"
- Chapter 10, "MPLS Application"
- Chapter 11, "10 Gb Ethernet IPv4/IPv6 Forwarding/Tunneling Application"
- Chapter 12, "Core Router Application"
- Chapter 13, "Dual OC-12 POS/Dual Gb Ethernet Forwarding Application for IXDP24X1"
- Chapter 14, "Quad Gigabit Ethernet Forwarding Application for IXDP24X1"
- Chapter 15, "ATM/Ethernet IPv4 Forwarding Application for IXDP24X1"
- Chapter 16, "POS/Ethernet IPv4 Forwarding Application for IXDP28x1"

1.3 Supported Applications

Table 1-1 describes the processor, platform, and operating system supported for the applications contained in the Intel® Internet Exchange Architecture (Intel® IXA) SDK.

Table 1-1. Supported Applications

Processor	Platform	Operating System	Application Name	Release 3.5 Revisions
2400	IXDP 2400 and simulation	VxWorks* and Linux*	OC-48 POS-IPV4 Forwarding Application	No change
2400	IXDP 2400 and simulation	VxWorks* and Linux*	4Gb Ethernet IPv4 Forwarding Application	No change
2400	IXDP 2400 and simulation	VxWorks*	OC-48 ATM AAL5 IPv4 Forwarding Application	Add core components
2400	IXDP 2400 and simulation	VxWorks* and Linux*	4Gb Ethernet IPv4/IPv6 Forwarding application	Add local stack support on Linux
2400	IXDP 2400 and simulation	VxWorks*	OC-48 DiffServ for POS IPv4 Forwarder Application	No change
2400	IXDP 2400 and simulation	VxWorks*	DiffServ for ATM	No change
2400	IXDP 2400 and simulation	VxWorks*	OC-48 POS MPLS IPv4 Forwarder Application	Add MPLS core components and exception handling
2400	IXDP 2400 and simulation	VxWorks*	10 Gb Ethernet IPv4 Forwarding Application	No change
2400	IXDP 2400 and simulation	VxWorks*	4xOC-12 POS ATM/DiffServ IPv4 Forwarder	No change
2400	IXDP 2400 and simulation	VxWorks*	4xOC-12 ATM AAL5 Forwarder	No change
2800	IXDP 2800 and simulation	VxWorks* and Linux*	OC-192 POS IPv6/IPv4 Forwarding and Tunneling Application	Add hardware support and support for Linux OS
2800	IXDP 2800 and simulation	VxWorks*	Core Router (OC-192 POS MPLS IPv4 Forwarder)	No change
2800	IXDP 2800 and simulation	VxWorks* and Linux*	10x1GbE IPv4/IPv6 Forwarding and Tunneling	New, includes Linux core components
2800	IXDP 2800 and simulation	VxWorks*	10GbE IPv4/IPv6 Tunneling	New

Table 1-1. Supported Applications (Continued)

Processor	Platform	Operating System	Application Name	Release 3.5 Revisions
2800	IXDP 2800 and simulation	VxWorks*	OC-192 POS IPv4 MPLS	New
2400	IXDP 2401 and simulation	VxWorks* and Linux*	Dual OC-12 POS/ Dual Gigabit Ethernet Forwarding Application	Add support for Linux OS
2400	IXDP 2401 and simulation	VxWorks* and Linux*	Quad Gigabit Ethernet IPv4 Forwarding Application	Add support for Linux OS
2400	IXDP 2401 and simulation	Linux*	ATM (AAL5) IPv4 Forwarding Application	New
2800	IXDP 28x1 and simulation	Linux*	POS/Ethernet IPv4 Forwarder Application	New

1.4 Other Sources of Information

This manual is part of the Intel® Internet Exchange Architecture (Intel® IXA) Software Development Kit (SDK) documentation set, which also includes the following manuals:

- *Intel® Internet Exchange Architecture Portability Framework Reference Manual*
- *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*
- *Intel® Internet Exchange Architecture Software Building Blocks Reference Manual*
- *Intel® Internet Exchange Architecture Software Building Blocks Developer's Manual*
- *Intel® Internet Exchange Architecture Software Development Kit Software Framework Getting Started Guide*

OC-48 POS IPv4 Forwarding Application 2

This chapter describes an IPv4 Forwarding application for Packet over SONET (POS) implemented on two half duplex Intel® IXP2400 Network Processors connected to a CSIX switch fabric. The chapter also provides a high-level design overview and lists the different software components used to build this application.

The application described in this chapter is supported on the Intel® IXDP2400 Advanced Development Platform.

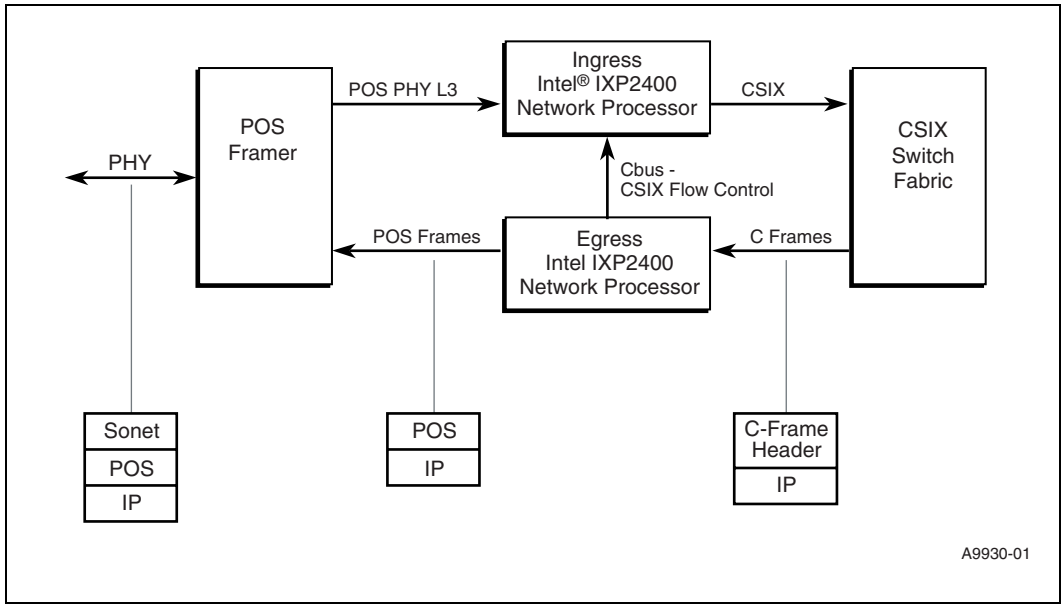
This chapter focuses only on the fast path or microengine components of the design. The XScale Core Components for this application are described in *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*.

This chapter describes the application in the context of a POS media interface. Chapter 3, "4Gb Ethernet IPv4 Forwarding Application" and Chapter 4, "OC-48 ATM IPv4 Forwarding Application" discuss the changes needed to support Ethernet and ATM.

2.1 Hardware Overview

Figure 2-1 shows two IXP2400 processors in a typical CSIX full duplex configuration. In this configuration, the two IXP2400 processors are identified as the ingress processor (receives from the Media interface and transmits to the CSIX Fabric) and the egress processor (receives from the CSIX Fabric and transmits to the Media interface).

Figure 2-1. Example Hardware Configuration for OC48-POS with CSIX Fabric



The Ingress IXP2400 receives POS frames that carry IPv4 datagrams. The frames are assembled into IPv4 packets and the Layer-2 (PPP) headers are removed. Based on the IPv4 header, a Longest Prefix Match (LPM) lookup is performed and the packets are segmented into CSIX C-Frames and transmitted to the CSIX fabric. The result of the LPM lookup determines which IXP2400 connected to the Fabric receives the packet, and which port on that IXP2400 the packet is transmitted on.

The Egress IXP2400 receives CSIX C-Frames from the fabric and reassembles these into IPv4 datagrams. The Layer-2 (PPP) headers are added and the packets are transmitted over the appropriate port.

2.2 Software Overview

Figure 2-2. Microblocks for an OC-48 POS IPv4 Forwarding Application

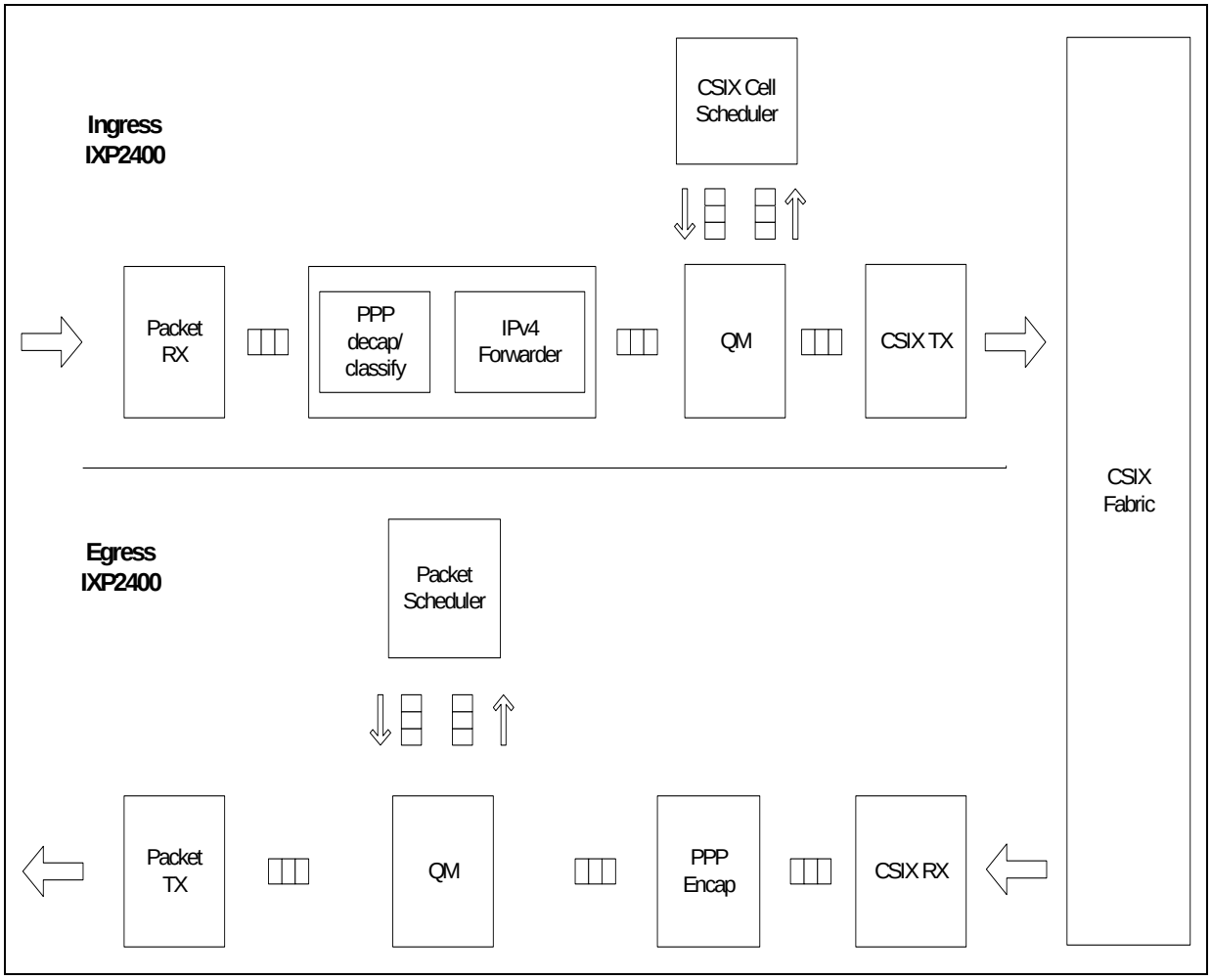


Figure 2-2 shows the microblocks needed to implement an OC-48 POS IPv4 Forwarding application. The design for this application is based on the guidelines specified in the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*. The driver microblocks (Receive, Transmit, Scheduler and Queue Manager) run on different microengines from the packet processing code. In this design, each driver block occupies an entire microengine. The packet processing blocks on the ingress IXP2400 include the IPv4 Forwarder and the PPP decapsulation/classify microblock. There are four microengines that run in parallel and execute the packet processing code. On the egress side, the only packet processing code is the PPP encapsulation block which runs on a single microengine.

2.2.1 Data Flow for the Ingress IXP2400

This section describes the data flow on the Ingress IXP2400:

2.2.1.1 Packet RX

The Packet Receive is a driver microblock that performs frame-reassembly on the mpackets coming in on the POS media interface. It reassembles and writes the packet data to a buffer in DRAM and queues the packet buffer handle on a microengine-microengine scratch ring for processing by the packet processing microengine. The Packet RX microblock also sets up per packet meta information (offset, size, etc.) which are passed on either in a descriptor in SRAM or in the microengine-microengine scratch ring itself. In this application, the packets reassembled are PPP frames containing IP datagrams. RFC 2615 defines the Packet Over SONET specification and refers to RFC 1661 (PPP) and RFC 1662 (PPP in HDLC-like framing). PPP framing including header validation, FCS generation and computation and byte stuffing are handled by the POS framer (IXF 6048).

The Packet RX microblock uses 8 threads on a single microengine, each of which handle one mpacket at a time. Up to 16 virtual ports are supported and the re-assembly context for all these ports is kept in local memory. To maintain packet sequencing, the threads execute in strict order.

Note: This microblock is written such that it supports up to 16 virtual ports, one or more of which may be unused. This allows the microblock to support different configurations such as Quad-OC12, 16 OC-3 or a single OC-48 port.

Since POS packets may be up to 9k bytes, some large packets may be stored in multiple buffers chained together as a link-list. The buffer handles for the first and last packet in the chain are queued in the scratch ring.

From the Packet RX block, the packet is passed on to an application specific system microblock (`DL_Sink[]`). This microblock checks if the packet has been marked to be dropped (`IX_DROP`) or sent to the XScale Core (`IX_EXCEPTION`). If not, it queues the packet buffer handle and associated packet meta data into the scratch ring for the next stage in the application.

2.2.1.2 PPP Decapsulation and Classify

The PPP decapsulation/classify microblock runs along with the IPv4 microblock on 4 microengines or 32 threads.

An application specific system source microblock on each thread dequeues packet buffer handles from the scratch ring. This source block (`DL_Source[]`) is a system microblock implicit in the dispatch loop. It reads in the packet meta information—that is, the packet descriptor, and populates the dispatch loop state. It also reads in 32 bytes of the packet header from DRAM into a header

cache maintained in transfer registers. Since it is important to maintain packet sequencing, the threads in the microblock execute in strict order to dequeue from the scratch ring. This implies that the first thread on microengine 1 dequeues the first packet, signals the next thread to perform the dequeue… etc. From this block, the packet goes to the PPP decapsulation/classify microblock.

The PPP decapsulation/classify microblock removes the layer-2 PPP header from the packet by updating the offset and size fields in the packet meta descriptor. Based on the PPP header, it also classifies the packet into IPv4, IPv6, PPP control packet (LCP, IPCP etc). If the packet is a PPP control packet, it is marked as an exception packet to be sent to the XScale Core (`IX_EXCEPTION`). Otherwise the packet is sent down the microengine stages for further processing. In this application, the dispatch loop silently drops packets classified as IPv6.

2.2.1.3 IPv4 Forwarder

The IPv4 forwarder microblock validates the IP header per RFC 1812. If the validity checks fail, then the packet is set up to be dropped as specified in Chapter 5, "Microblocks" of the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*. Otherwise, a Longest Prefix Match (LPM) is performed on the IPv4 header. The result is an IPv4 Next Hop ID, a fabric blade ID (identifying a unique IXP2400 on the fabric) and an output port identifying the output port on the Egress IXP2400. The Next Hop ID is passed over the CSIX fabric to an Egress IXP2400 where it is used to look up information about the Layer-2 header to be prepended to the packet buffer. The output port is also passed over the CSIX fabric to the Egress IXP2400 and is used to transmit over the appropriate port. All three fields are stored in the packet meta data—that is, the packet descriptor.

If no match is found, then the packet is set up to be sent up to the XScale core for further processing. Packets are also sent to the core in a number of other cases, for example, when the packet is destined for a local interface or is to be fragmented.

From the IPv4 forwarder block, the packet is passed on to an application specific system microblock (`DL_QM_Sink[]`). This microblock checks if the packet is to be dropped or sent to the XScale Core. If not, it sends an enqueue request to the Queue Manager over a scratch ring. `DL_QM_Sink[]` also writes the cached packet header to DRAM and the packet meta information to SRAM.

2.2.1.4 Cell Based Queue Manager (Cell QM)

The Queue Manager is a driver microblock that is implemented as a single microblock that runs on a single microengine. Since this is the only code running on the microengine and it does not really process packets, there is no need for a dispatch loop.

The QM is responsible for performing enqueue and dequeue operations on the transmit queues which are implemented using the hardware SRAM link lists. It accepts enqueue requests from the functional pipeline via a scratch ring. The enqueue requests are on a per-packet basis. The dequeue requests come from the transmit scheduler microengine on a per-cell basis where a cell is a CSIX cframe. Whenever an enqueue results in the queue state going from empty to non-empty or a dequeue operation results in the queue state going from non-empty to empty, the Queue Manager sends a message to the transmit scheduler via a Next Neighbor Ring. Also after every dequeue, the QM passes a transmit request via a scratch ring to the CSIX TX microblock.

The threads on the QM microengine execute in strict order using local inter-thread signaling. SRAM Queue Array entries are cached in the SRAM Controller and the CAM is used for managing the tags for these. To maintain coherence among threads, folding is used.

2.2.1.5 CSIX Scheduler

The CSIX scheduler is a driver microblock that runs on a single microengine. Since this is the only code running on the microengine and it does not process packets, there is no need for a dispatch loop.

The CSIX scheduler schedules packets to be transmitted to the CSIX fabric. The scheduling algorithm implemented is Round Robin among the ports on the fabric and optionally Weighted Round Robin among the queues on a port. Since this is not a QoS application and there is only one queue per port, the Weighted Round Robin scheduling may either be compiled out or made to degenerate to round robin scheduling. Other applications—for example, IP DiffServ, may use the WRR functionality. The scheduling and transmit is done a cframe at a time.

The CSIX scheduler handles

- Flow control messages from the fabric

  These messages are sent by the fabric to the Egress IXP2400, which sends them on the c-bus to the Ingress IXP2400. If the fabric asserts Xoff on a particular VoQ (Virtual Output Queue), the scheduler stops scheduling for the queue.

- Queue transition messages from the queue manager

  A queue is scheduled only if there is data in the queue.

- MSF Transmit State Machine

  The scheduler monitors how many packet cframes are in the pipeline and if it exceeds a certain threshold, it stops scheduling.

For both the VoQ status and the transmit queue status, the scheduler keeps hierarchical bit vectors and uses the MEv2 FFS (Find First Bit) instruction to scan them efficiently. During each loop, the scheduler

- Checks if the TX pipeline is within a pre configure threshold
- Picking up from where it left off in the last iteration it finds the next bit set and determines which queue to schedule.
- It then sends a dequeue message to the Queue Manager to dequeue the head of that queue. The Queue Manager dequeues a cell (cframe) from the head of the queue and sends a transmit request on a scratch ring to the CSIX TX microblock.

2.2.1.6 CSIX TX

CSIX Transmit is a driver microblock that runs on a single microengine. It receives transmit messages from the queue manager. With each transmit request, the microblock moves a cframe into a TBUF, which is then transmitted into the fabric by the MSF Transmit State Machine.

Every request has an associated packet, which is being segmented into cframes. The associated segmentation state for the packet and the packet metadata is cached in local memory and is looked up using the CAM. The TX microblock adds the CSIX header onto the cframe along with the packet data. Along with the CSIX header, a Traffic Manager (TM) header is also added per cframe carrying extra information (destination Layer-2 port ID, input blade ID, sequence number, next-hop ID, etc.) about the packet to be passed to the Egress IXP2400. In addition, the flow ID, class ID, input port, and some other fields from the metadata are passed along to the Egress IXP2400 using a per-packet header pre-pended to the start of the first c-frame of each packet.

As in other drive microblocks, the threads use folding and execute in strict order. If an entire buffer for a packet has been transmitted, then the buffer is freed.

2.2.2 Data Flow for the Egress IXP2400

2.2.2.1 CSIX RX

The CSIX RX is a driver microblock that runs on a single microengine. It receives c-frames from a CSIX fabric and reassembles them into IP packets. A key difference between the CSIX receive and the POS receive microblock is that while the Packet RX block supports only 16 virtual ports, the CSIX RX block supports up to 64k VOQ's. This implies that the reassembly contexts (RXC) are stored in SRAM. The folding technique is used to optimize the read modify write of the context. Sixteen contexts are cached in local memory at any time and the CAM is used to lookup the context. The ingress blade id and the QoS class are used to uniquely identify a context and are used as a key for the CAM lookup.

Since the packets being reassembled may be up to 9k bytes, some large packets may be stored in multiple buffers chained together as a link-list. The buffer handles for the first and last packet in the chain are queued in the scratch ring. The CSIX RX microblock also sets up packet meta information (offset, size etc.) which are passed on to the packet processing microengines either in SRAM or in the scratch ring itself.

2.2.2.2 PPP Encapsulation

This block adds the layer-2 PPP header to the packet and enqueues it to the next stage of the pipeline. If the next hop id in the packet meta data is set to an invalid value (-1) then the block assumes that the PPP header has already been added to the packet and simply enqueues it to the next stage of the pipeline.

2.2.2.3 Packet Based Queue Manager

This block is virtually identical to the Cell Based Queue Manager except that it dequeues packets. The SRAM Q-Array hardware is programmed in packet mode and ignores the cell count field in the buffer handle. The cell count field may be used to store an approximation of the length of the packet for DRR scheduling. Another key difference between the cell based and packet based queue manager is that the packet based queue manager returns a dequeue response message to the scheduler for every dequeue request. This dequeue response contains the packet length which is needed by the scheduler for implementing the DRR algorithm. The dequeue response message is combined with the enqueue/dequeue transition messages and is returned on the same next neighbor ring.

On the ingress side (Cell QM), the scheduler does not need the packet length. Therefore a message is only sent from the QM to the scheduler in case of a queue transition or if the dequeue was invalid.

2.2.2.4 Egress Packet WRR/DRR Scheduler

The Egress scheduler schedules POS packets to be transmitted over the POS interface. The key difference between the ingress and Egress IXP2400’s is that the egress scheduler is a packet-based scheduler as opposed to the cell (i.e., c-frame) based scheduler on the ingress side. Also there are no flow control messages to be processed from the fabric.

The Egress scheduler implements Weighted Round Robin (WRR) scheduling among the ports and optionally DRR (Deficit Round Robin) scheduling among the queues on a port. Since this is not a QoS application and there is only one queue per port, the DRR is compiled out or made to degenerate to round robin.

Using the Weighted Robin algorithm on the 16 virtual ports allows us the flexibility to support a number of different configurations such 16-OC3, 3 OC-12, and 4 OC-3, etc. The weights on the ports are adjusted according to the data rate sustained on that port.

To prevent head-of-line blocking, the scheduler with the help of feedback from the Packet TX block keeps track of the number of packets in flight (scheduled, but not transmitted) for each port. If this number exceeds a specified limit, then it stops scheduling on that port.

2.2.2.5 Packet TX

The Packet TX microblock transmits packets over the media interface. It segments a packet into mpackets and moves them into TBUFS for the MSF state machine to transmit. This is similar to the CSIX TX microblock except that instead of adding the CSIX header, the Packet TX microblock assumes that the layer-2 header is already prepended to the start of the packet by a previous stage of the packet processing pipeline. Also while the CSIX TX block receives a transmit request for every cframe, the Packet TX microblock receives a transmit request for the entire packet.

The MPHY-16 Packet TX microblock supports up to 16 virtual ports. The transmit context for all of these are kept in local memory. Therefore the CAM is not required. The Packet TX microblock monitors the MSF to see if the TBUF threshold for a specific port has been exceeded. If so it stops transmitting on that port and any requests to transmit packets on that port are queued up in local memory.

The Packet TX microblock periodically updates the scheduler with information about how many packets have been transmitted. If the packets in flight for a particular port (packets scheduled but not transmitted) exceed a certain limit (which depends on the bandwidth supported by that port), then the scheduler stops scheduling any more packets for the port. This combination of queuing packets in local memory and keeping track of the packets in flight helps prevent 'head of line blocking'.

One thing to note is that the design is much simpler for the case where only a single OC-48 port (SPHY mode) needs to be supported. This is because there are no head of line blocking issues and packets needed not be queued in local memory. The same applies for a quad OC12 design (MPHY-4 mode) where we can avoid head of line blocking issues by using four different scratch rings (1 per port) and allocate two microengine threads for each port.

Another assumption made in this design is that the output port for egress is found via the IPv4 lookup performed on the ingress side. A different approach is to use the next hop id and do a lookup on the egress side to find out the output port number.

The Packet TX microblock can be used to support the following configurations:

- SPHY 1x32 (single port OC-48)
- MPHY-4 (or SPHY 4x8—four port OC-12)
- MPHY-16 (up to 16 virtual ports)

In the single port or four port configuration, the Packet TX microblock runs on a single microengine, while in the MPHY-16 mode it runs on two microengines.

2.2.3 Dispatch Loops / Microblock Groups

There are two dispatch loops (microblock groups) on the ingress pipeline. For more information on dispatch loops, see Chapter 6, "Dispatch Loop" in the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*.

- Dispatch Loop for the Packet Frame Reassembly Stage (Figure 2-3)
- Dispatch Loop for the IPv4 Forwarder functional pipeline (Figure 2-4)

The QM, Scheduler, and CSIX TX blocks don't use a dispatch loop, though they still use the dispatch loop macros where required.

Figure 2-3. Dispatch Loop for the Packet Frame Reassembly Stage

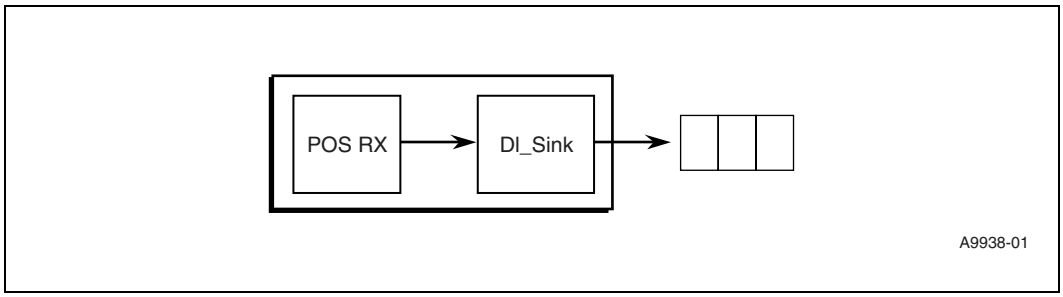
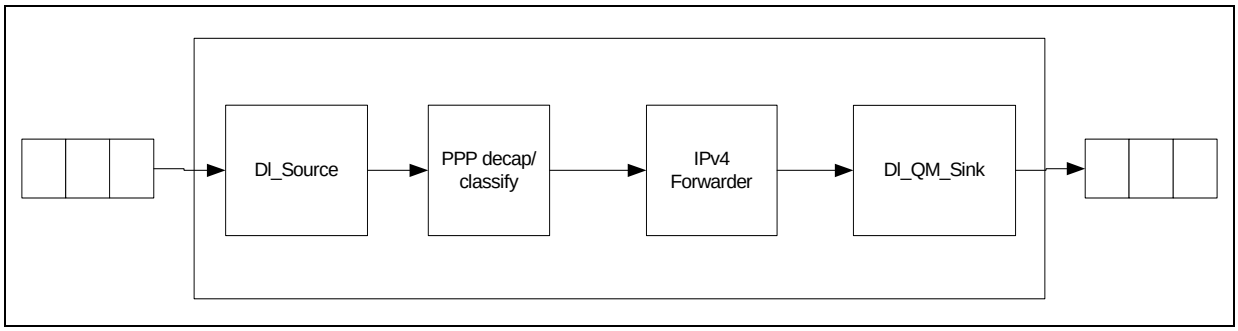


Figure 2-4. Dispatch Loop for the IPv4 Functional Pipeline



Note that the system microblocks `dl_source`, `dl_sink`, `dl_qm_sink`, etc are application specific. They may be change for different packet processing pipelines.

There are two dispatch loops on the egress pipeline

- Dispatch Loop for the CSIX RX Reassembly stage (Figure 2-5)
- Dispatch Loop for the PPP encapsulation stage (Figure 2-6)

Figure 2-5. Dispatch Loop for CSIX Reassembly Stage

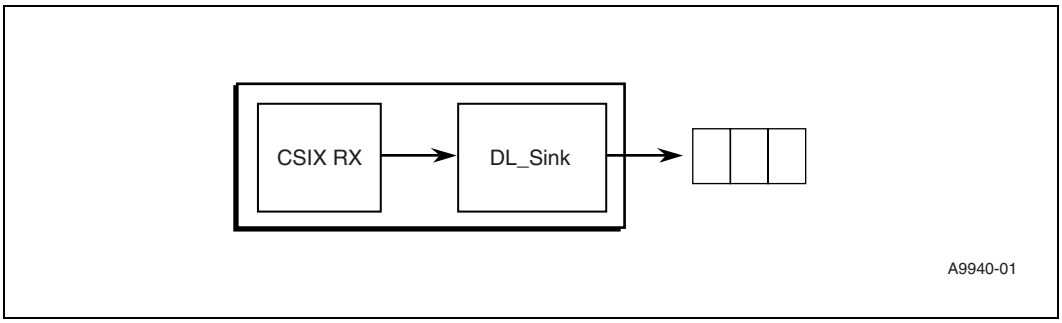
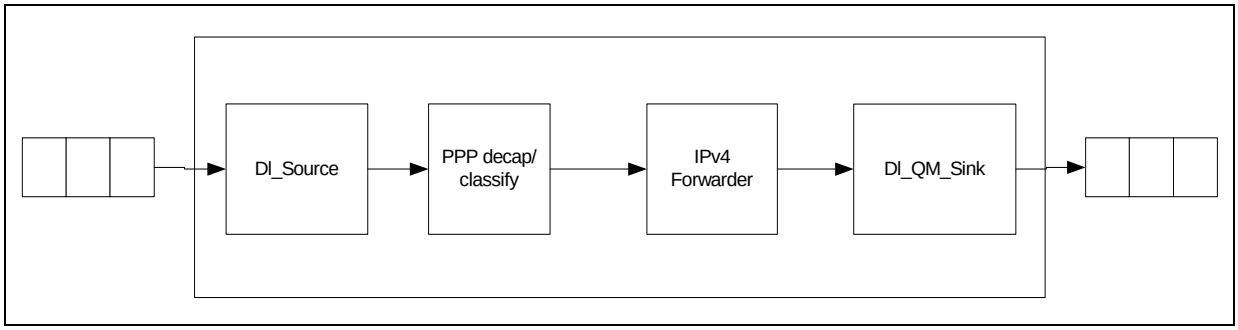


Figure 2-6. Dispatch Loop for POS Transmit Stage



2.3 Performance Characterization

The IXP2400 operates at 600 MHz. For a min POS packet of 49B, the packet inter-arrival time at OC-48 line rate is 97 microengine cycles. In order to maintain line rate for min packets, each stage of the pipeline cannot exceed this budget. In other words, each stage of the pipeline needs to retire a packet every 97 cycles.

Table 2-1 summarizes the performance analysis for the POS pipeline.

Table 2-1. Performance Characterization for the POS Pipeline

Parameter	Value
OC-48 line rate assuming 3% SONET overhead	2.408 Gigabits/sec
Min POS packet size	49 bytes (40 byte TCP/IP, 2 bytes Address and Control, 2 byte PPP header, 4 byte FCS and 1 byte flag)
Packet Throughput for min packets	6.14 million packets/sec = (2.408 / (49*8)) * (10**9)
IXP2400 clock frequency	600 MHZ
Inter-packet arrival time for min packets	600/6.14 = 97.7 cycles
Compute cycles per packet for a single microengine	97

Table 2-1. Performance Characterization for the POS Pipeline (Continued)

Parameter	Value
Latency per packet for a single microengine	97 * 8
Compute cycles per packet for n microengines running in parallel	97*n
Latency per packet for a n microengines running in parallel	97*8*n

2.4 Ingress System Resource Allocation

Table 2-2 shows the system resources mapped for the Ingress IXP2400. This mapping reflects the system defaults and may be changed. The allocation of microengines is done such that it optimizes the performance of this specific application and may be changed for other applications.

Table 2-2. System Resources Mapped for the Ingress IXP2400

Microblock	ME #	Communication Mechanism with previous stage
Packet RX	ME0	Auto-push status from MSF
IPv4 Forwarder + Layer2 decapsulation/Classify	ME1, ME2, M5, M6	Scratch ring
Queue Manager	ME3	Scratch ring
CSIX Scheduler	ME4	Next neighbor + Scratch ring
CSIX TX	ME7	Scratch ring

The physical assignment of function to microengine is important since it not only affects when the next neighbor registers and signaling can be utilized, but it also affects the utilization of the internal Command bus and S-Push/Pull buses. Since ME0-ME3 belong to Microengine Cluster 0 and ME4-ME7 belong to Microengine Cluster 1, this assignment attempts to balance the usage of the Command bus and S-Push/Pull buses across the two clusters.

IXP2400 supports two SRAM channels and one DRAM channel. Table 2-3 shows the SRAM, DRAM and scratch utilized for this application. These values are defined in a system header file `dl_system.h` and may be changed as required.

Table 2-3. SRAM, DRAM and Scratch Utilization for Ingress System Resources

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Buffer Descriptors	32	32k (In simulation, we use only 320 buffers)	1 MB		
Buffers	2048	32k		64 MB	
Queue Descriptors	16	1024 (1 per VOQ)	16K		
CSIX TX contexts	32	1024 (1 per VOQ)	32k		

Table 2-3. SRAM, DRAM and Scratch Utilization for Ingress System Resources (Continued)

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Trie Table	64 (The root Trie table requires at least 257k to support hi64k and hi256 tables. In addition each node requires 64 bytes. These nodes are added as needed)	See note in previous column. Assuming 256k routes, approximately 128k nodes are needed	8MB		
Route Table (Next Hop Information)	16	Assuming 4k next hops	64k		
IPv4 statistics	4	16			64
Packet RX statistics	4	16*16	1024		
IPv4 Directed Broadcast Table	32	256	8k		
Ring from Packet RX to packet processing pipeline (IPv4+Layer2 Decap/Classify)	20	4k/20			4k
IPv4 to QM ring	12	2k/12			2k
Scheduler to QM	4	512			2k
QM to CSIX TX	8	256			2k
QM Q-Array entries	N/A	16			
Buffer Free list Q-Array entry	N/A	4			

2.5 Egress System Resource Allocation

Table 2-4 shows the system resources allocated for the Egress IXP2400.

Table 2-4. System Resources Allocated for the Egress IXP2400

Microblock	ME #	Communication Mechanism with previous stage
CSIX RX	ME0	Auto-push status from MSF
Packet TX	ME4, ME5 (For SPHY 1x32, one microengine is sufficient. For MPHY-16 designs two microengines are needed)	Scratch ring
Layer-2 Encapsulation	ME3	Scratch ring
Egress QM	ME1	Scratch Ring
Egress Scheduler	ME2	Next neighbor + Scratch ring
Unused (available headroom)	M6, ME7	N/A

The mapping of networking functions on to the microengines shows that six microengines are used to perform the fast path processing for this application. Additional functionality required by customers can be mapped on to the remaining microengines.

Table 2-5 shows how the SRAM, DRAM and scratch are utilized for this application. These values are defined in a system header file `dl_system.h` and may be changed as required.

Table 2-5. SRAM, DRAM and Scratch Utilization for Egress System Resources

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM Used	Total Scratch used
Buffer Descriptors	32	32k (In simulation we use only 320 buffers)	1 MB		
Queue Descriptors	16	256 (16 ports x 16 classes per port)	4k		
CSIX RX Reassembly contexts	32	1024	32k		
Buffers	2048	32k		64 MB	
CSIX RX to Layer-2 Encap ring	12	512/3 (the size of the ring is 512 long words, but each entry enqueued uses 3 long words. Therefore the total number of entries is 512/3 = 170)			2k
Layer-2 Encap to QM ring	12	512/3			2k
Scheduler to QM ring	4	512			2k
QM to POS TX	4	512512			2k2k
QM Q-Array entries	N/A	16			
Buffer Free list Q-Array entry	N/A	4			

2.6 Interfaces Between the Various Microblocks

This section describes the interfaces between the different microblocks in the ingress and egress processors for this application.

In most of the messages, there is a valid bit is used to prevent a value of zero from being enqueued on the scratch ring. Zero is used to detect a case where the scratch ring is empty. So the valid bit helps us distinguish between a zero value that was actually enqueued versus a case where the ring is empty.

2.6.1 Packet RX and Packet Processing Microengines

The interface between the Packet Receive microblock and the Packet Processing Microengines (IPv4 Forwarder + PPP decap) is a scratch ring. Table 2-6 describes each entry in the scratch ring— which is five words.

Table 2-6. Five-Word Entry in Scratch Ring (IPv4 Forwarder + PPP Decap)

LW	Bits	Size	Field	Description
0	31:0	32	dl_buffer_handle	Buffer Handle for the SOP Descriptor
1	32:0	32	dl_eop_buffer_handle	Buffer Handle for the EOP Descriptor
2	31:16	16	buffer_size	Buffer size in bytes
	15:0	16	offset	Offset of the start of data in the buffer in bytes
3	31:28	16	packet_size	Total packet size across buffers
	15:12	4	free_list_id	Free list ID for buffer
	11:8	4	rx_stat	Receive Status Flag
	7:0	8	header_type	Type of header at offset bytes into the packet
4	31:16	16	input_port	Input port on ingress processor
4	15:0	16	reserved	Reserved

2.6.2 Packet Processing Microengines and Cell Queue Manager

The interface between Packet Processing Microengines (IPv4 Forwarder + PPP decap) and Cell Queue manager is a scratch ring. Table 2-7 describes each entry in the scratch ring—which is three words.:

Table 2-7. Three-Word Entry in Scratch Ring (IPv4 Forwarder + PPP Decap)

LW	Bits	Size	Field	Description
0	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
1	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)
2	31	1	Valid Bit	Must be 1
2	30:16	15	Reserved	Reserved
2	0:15	16	Queue Number	Queue Number

2.6.3 Cell Queue Manager and CSIX Scheduler

Table 2-8 describes the CSIX scheduler issued dequeue requests to the Cell based Queue Manager via a scratch ring.

Table 2-8. Dequeue Requests via the Scratch Ring

LW	Bits	Size	Field	Description
0	31	1	Valid Bit	Must be 1
0	30:16	15	Reserved	Reserved
0	0:15	16	Queue Number	Queue Number

Table 2-9 shows the Queue Transition Messages sent by the Queue Manager to the scheduler via a Next Neighbor Ring.

Table 2-9. Queue Transition Messages Sent by the Queue Manager

LW	Bits	Size	Field	Description
0	31	1	Valid Bit	Must be 1
	30	1	Enqueue Transition	Notification that queue has gone from empty to non-empty
	29:16	1	Reserved	Reserved
	28:18	11	Packet cell count	Unused for CSIX
	17:16	2	Reserved	Reserved
	15:0	16	Queue Number	Queue Number that was enqueued (Only 10 bits are used for CSIX)
1	31	1	Valid Bit	Must be 1
	30	1	Dequeue Transition	Notification that queue has gone from non-empty to empty
	29	1	Invalid Dequeue	If set, then dequeue request to an invalid queue was made
	28:16	13	Packet size	Unused for CSIX
	15:0	16	Queue Number	Queue Number that was dequeued (Only 10 bits are used for CSIX

2.6.4 Cell Queue Manager and CSIX TX

The interface between the Cell based Queue Manager and the CSIX TX block is a scratch ring. Table 2-10 describes each entry in the scratch ring—which is two words.

Table 2-10. Two-Word Entry in Scratch Ring

LW	Bits	Size	Field	Description
0	31	1	Valid Bit	Must be 1
0	30:16	15	Reserved	Reserved
0	15:0	16	Queue Number	Queue Number
1	31:0	32	Buffer Handle	Buffer Handle currently being transmitted for queue

2.6.5 CSIX RX and PPP Encap

The interface between CSIX RX and PPP Encap is a scratch ring. Table 2-11 describes each entry in the scratch ring—which is two words.

Table 2-11. Three-Word Entry in Scratch Ring (CSIX and PPP Encap)

LW	Bits	Size	Field	Description
0	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
1	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)
2	31	1	Valid Bit	Must be 1
2	30:16	15	Reserved	Reserved
2	0:15	16	Queue Number	Queue Number

2.6.6 PPP Encap and Packet Queue Manager

Table 2-12 shows the scratch ring interface between the PPP Encap and Packet Queue Manager.

Table 2-12. Scratch Ring Interface between PPP Encap and Packet Queue Manager

LW	Bits	Size	Field	Description
0	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
1	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)
2	31	1	Valid Bit	Must be 1
2	30:16	15	Reserved	Reserved
2	0:15	16	Queue Number	Queue Number

2.6.7 Packet Queue Manager and Scheduler

The interface between the Packet based Queue Manager and the POS/Ethernet Scheduler is a Next Neighbor Ring.

The message format is identical to the interface between the Cell Queue Manager and the CSIX Scheduler except that an additional word containing the packet length is sent. The one difference is that while the Cell Queue Manager sends a message to the scheduler only on an enqueue/dequeue transition or in the case of an invalid dequeue, the Packet Queue Manager sends a dequeue response (combined with the transition messages) on every dequeue request. In the case where there is only an enqueue transition (no dequeue request was sent by the scheduler), the packet size is set to 0 by the queue manager.

Table 2-13. Queue Transition Messages Sent by the Packet Queue Manager

LW	Bits	Size	Field	Description
0	31	1	Valid Bit	Must be 1
	30	1	Enqueue Transition	Notification that queue has gone from empty to non-empty
	29:16	1	Reserved	Reserved

Table 2-13. Queue Transition Messages Sent by the Packet Queue Manager (Continued)

LW	Bits	Size	Field	Description
	28:18	11	Packet cell count	Unused for POS/Ethernet
	17:16	2	Reserved	Reserved
	15:0	16	Queue Number	Queue Number that was enqueued (Only 8 bits are used for POS/Ethernet)
1	31	1	Valid Bit	Must be 1
	30	1	Dequeue Transition	Notification that queue has gone from non-empty to empty
	29	1	Invalid Dequeue	If set, then dequeue request to an invalid queue was made
	28:16	13	Packet size	Size of the packet in units of 128 bytes (Only 7 bits are used)
	15:0	16	Queue Number	Queue Number that was dequeued (Only 8 bits are used for POS/Ethernet)

2.6.8 Packet Queue Manager and Packet TX

The interface between the Egress Queue Manager and the Packet Transmit for the POS and Ethernet applications is a scratch ring. Table 2-11 describes each entry in the scratch ring—which is one word.

Table 2-14. One-word Scratch Ring Entry

LW	Bits	Size	Description
0	31:31	1	Valid bit
	30:28	3	Reserved
	27:24	4	Port number
	23:0	24	Pointer to SOP buffer descriptor in SRAM in long words (Same as bits 0:23 of buffer handle)

2.7 Core Components

The following sections list the core components used by this application and describe how exception packets are handled.

2.7.1 Ingress Core Components

The core components that run on the Ingress side are POS Rx, IPv4 Forwarder, Stack Driver, Queue Manager, CSIX Tx, and Scheduler. In addition, there are several libraries that are required for the functioning of these core components: the Route Table Manager, Fragmentation, and Message Support libraries. There is another component called System Application that plays the role of a system designer. For details on these core components, refer to the *Intel® Internet Exchange Architecture Software Building Blocks Developer's Manual*.

2.7.2 Egress Core Components

The core components that run on the Egress side are POS Tx, Scheduler, CSIX Rx, and Queue Manager. The library that is required on the Egress side is the Message Support library. There is another component called System Application that plays the role of a system designer. For details on these core components, refer to the *Intel® Internet Exchange Architecture Software Building Blocks Developer's Manual*.

2.7.3 Exception Path Processing

Non-IP exception packets are delivered to the POS Receive core component. All such packets will be sent to a component output defined in the file `bindings.h`. By default, this output is bound to IX_DROP. Any other component or application that needs these packets can redefine communication ID for the output. Exception IP packets are delivered to IPv4 Forwarder core component. If the packet is for local delivery, it gets sent to the Stack Driver and then to the local or remote control plane.

The IPv4 Forwarder core component processes all other IPv4 packets and either forwards them to the Queue Manager core component, or discards them. In addition, the IPv4 Forwarder core component generates ICMP messages.

Outbound packets are delivered to the microblocks through the Queue Manager core component.

4Gb Ethernet IPv4 Forwarding Application 3

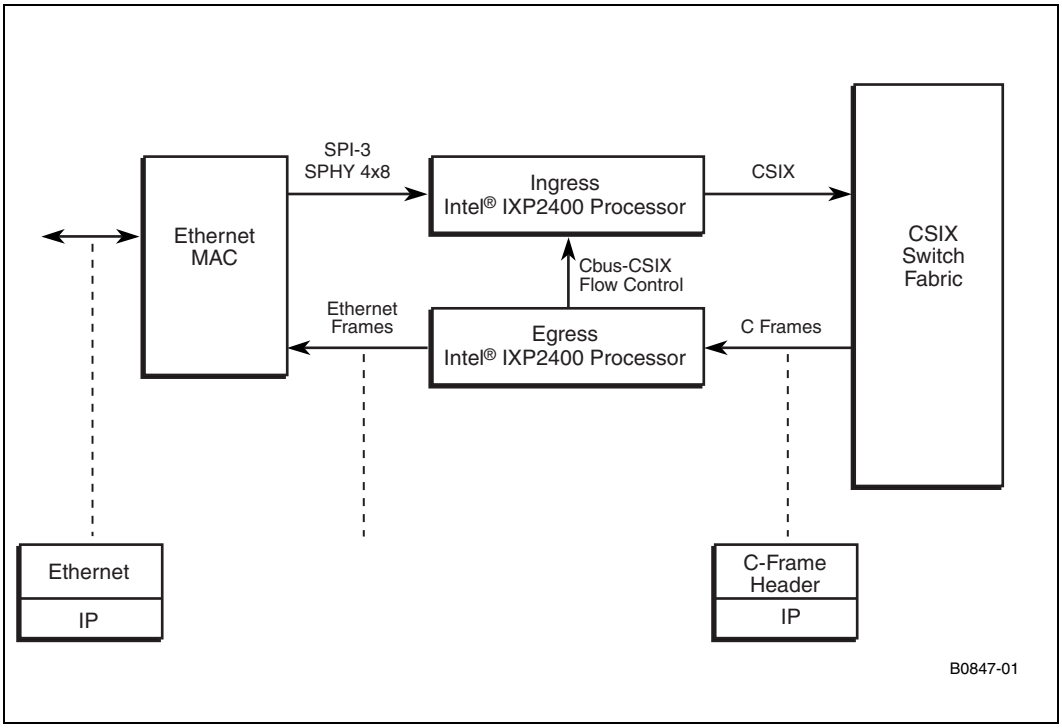
This section describes the design of an IPv4 Forwarding application using the Intel® IXP2400 Network Processor. Two half-duplex IXP2400 processors are used to implement a 4GB Ethernet line card that interfaces to a CSIX switch fabric. This section provides a high-level design overview and lists the different software components used to build this application. It focuses only on the fast path or microengine components of the design. The Intel XScale® core components for this application are described in *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*.

The application described in this chapter is supported on the Intel® IXDP2400 Advanced Development Platform.

3.1 Hardware Overview

Figure 3-1 shows two IXP2400 processors in a typical CSIX full duplex configuration. In this configuration, the two processors are identified as the ingress processor (receives from the Media interface and transmits to the CSIX Fabric) and the egress processor (receives from the CSIX Fabric and transmits to the Media interface). The hardware is configured in SPHY 4x8 mode. Up to 4 Gigabit Ethernet ports are supported—one port per 8-bit wide SPHY channel.

Figure 3-1. Example Hardware Configuration for 4x1 Gigabit Ethernet with CSIX Fabric



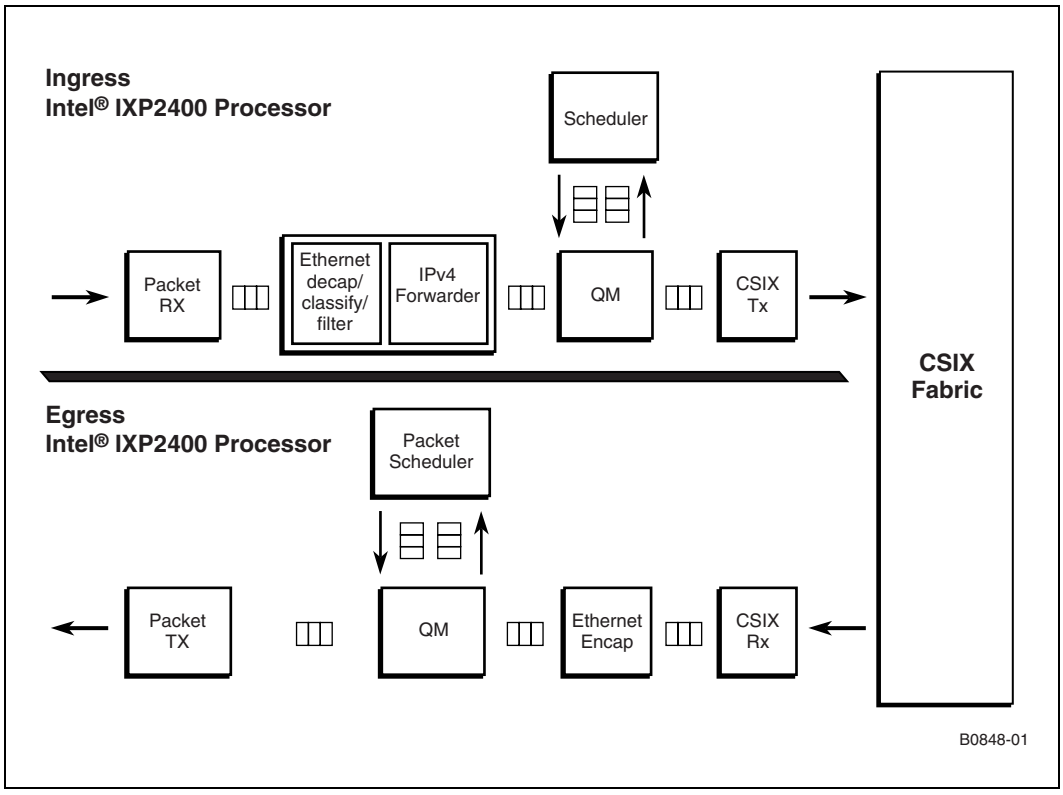
The Ingress IXP2400 receives Ethernet frames that carry IPv4 datagrams. The frames are assembled into IPv4 packets and the Layer-2 (Ethernet) headers are removed. Based on the IPv4 header, a Longest Prefix Match (LPM) lookup is performed and the packets are segmented into CSIX C-Frames and transmitted to the CSIX fabric. The result of the LPM lookup determines which IXP2400 connected to the Fabric receives the packet, and which port on that IXP2400 the packet is transmitted on.

The Egress IXP2400 receives CSIX C-Frames from the fabric and reassembles these into IPv4 datagrams. The Layer-2 (Ethernet) headers are added and the packets are transmitted over the appropriate port.

3.2 Software Overview

Figure 3-2 shows the microblocks needed to implement an Ethernet IPv4 Forwarding application. The design for this application is based on the guidelines specified by the IXA Portability Framework—*Intel® Internet Exchange Architecture Portability Framework Developer's Manual*. The driver microblocks (Receive, Transmit, Scheduler and QM) run on different microengines from the packet processing code. In this design, each driver block occupies an entire microengine. The packet processing blocks on the ingress IXP2400 include the IPv4 Forwarder and the Ethernet decapsulation/classify microblock. There are four microengines that run in parallel and execute the packet processing code. On the egress side, the only packet processing code is the Ethernet Encapsulation/ARP block which runs on a single microengine.

Figure 3-2. Software Components for IPv4 Forwarding Application for Ethernet



3.2.1 Data Flow for the Ingress IXP2400

This section describes the data flow on the Ingress IXP2400:

3.2.1.1 Packet RX

This block is identical to the Section 2.2.1.1, “Packet RX” on page 25 except that it sets the header type field in the packet meta data to Ethernet.

3.2.1.2 Ethernet Decapsulation/Classify/Filter

The Ethernet decapsulation/classify/filter microblock runs in a functional pipeline with the IPv4 microblock on 4 microengines or 32 threads.

This microblock removes the layer-2 Ethernet header from the packet by updating the offset and size fields in the packet meta data. It also implements MAC filtering based on the destination MAC address in the Ethernet header. Based on this filtering, the packet may be dropped.

This microblock also classifies the packet into IPv4, IPv6, MPLS, ARP etc. If the packet is an ARP packet, it is marked as an exception packet to be sent to the Intel XScale® core (IX_EXCEPTION). Otherwise the packet is sent down the microengine pipeline for further processing. In this application, the dispatch loop silently drops packets classified as IPv6 or MPLS.

3.2.1.3 IPv4 Forwarder

This block is identical to the block described in Section 2.2.1.3, “IPv4 Forwarder” on page 26.

3.2.1.4 Cell Based Queue Manager (Cell QM)

This block is identical to the block described in Section 2.2.1.4, “Cell Based Queue Manager (Cell QM)” on page 26.

3.2.1.5 CSIX Scheduler

This block is identical to the block described in Section 2.2.1.5, “CSIX Scheduler” on page 27.

3.2.1.6 CSIX TX

This block is identical to the block described in Section 2.2.1.6, “CSIX TX” on page 27.

3.2.2 Data Flow for the Egress IXP2400

This section describes the data flow for the Egress IXP2400.

3.2.2.1 CSIX RX

This block is identical to the block described in Section 2.2.2.1, “CSIX RX” on page 28.

3.2.2.2 Ethernet Encapsulation

This block adds the layer-2 Ethernet header to the packet and enqueues it to the next stage of the pipeline. It uses the next hop id as an index into a table with layer-2 header information. If the layer-2 header is not found, the packet is enqueued to be processed by the Intel XScale® core. ARP Processing is handled by the Intel XScale® core application code. If the next hop id is set to an invalid value (-1), the block assumes that the layer-2 header has already been added to the packet and sends it to the next stage of the pipeline.

3.2.2.3 Packet Based Queue Manager (Packet QM)

This block is identical to the block described in Section 2.2.2.3, “Packet Based Queue Manager” on page 28.

3.2.2.4 Egress Scheduler

This block is identical to the block described in Section 2.2.2.4, “Egress Packet WRR/DRR Scheduler” on page 28.

3.2.2.5 Packet TX

This block is identical to the block described in Section 2.2.2.5, “Packet TX” on page 29.

3.2.3 Dispatch Loops / Microblock Groups

There are two dispatch loops (microblock groups) on the ingress pipeline

- Dispatch Loop for the Packet Frame Reassembly Stage (Figure 3-3)
- Dispatch Loop for the IPv4 Forwarder functional pipeline (Figure 3-4)

The QM, Scheduler and CSIX TX blocks don't use a dispatch loop (they still use the dispatch loop macros where required).

Figure 3-3. Dispatch Loop for the Packet Frame Reassembly Stage

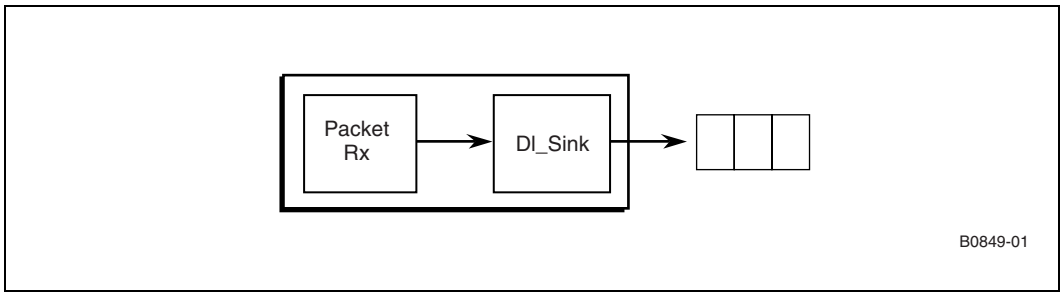
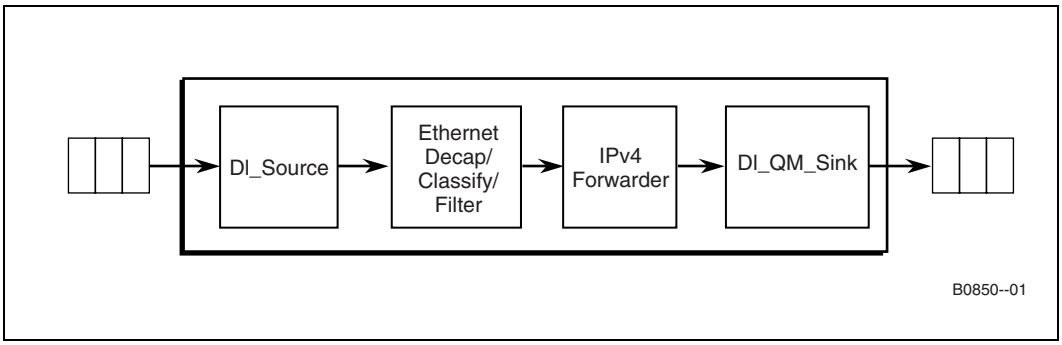


Figure 3-4. Dispatch Loop for the IPv4 Functional Pipeline



Note that the system microblocks dl_source, dl_sink, dl_qm_sink etc are application specific. They may be changed for different packet processing pipelines.

There are two dispatch loops (microblock groups) on the egress pipeline

- Dispatch Loop for the CSIX RX Reassembly stage (Figure 3-5)
- Dispatch Loop for the Ethernet encapsulation stage (Figure 3-6)

Figure 3-5. Dispatch Loop for CSIX Reassembly Stage

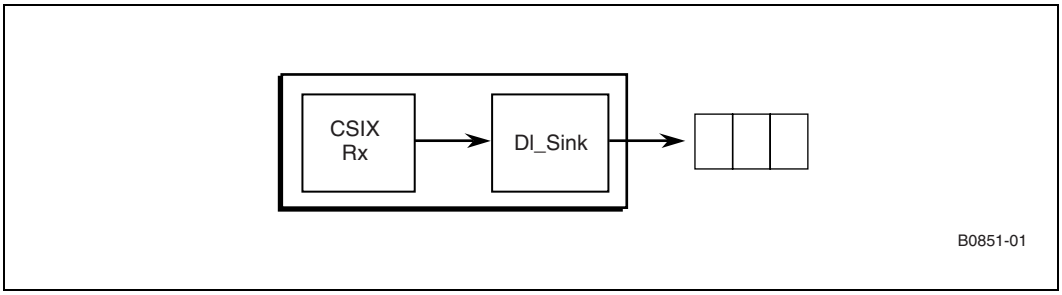
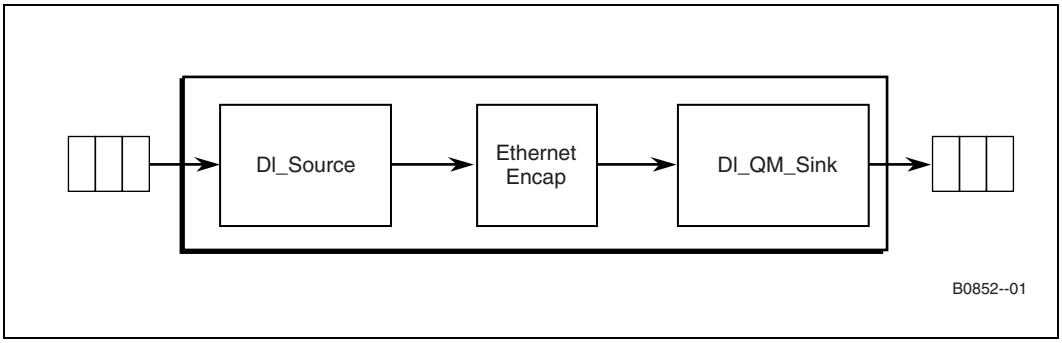


Figure 3-6. Dispatch Loop for Ethernet Encapsulation Stage



3.2.4 Performance Characterization

The IXP2400 operates at 600 MHz. For a min Ethernet packet of 64B, the packet inter-arrival time at 4 Gbps line rate is 100 microengine cycles. In order to maintain line rate for min packets, each stage of the pipeline cannot exceed this budget. In other words, each stage of the pipeline needs to retire a packet every 100 cycles. Table 3-1 summarizes the performance analysis for the Ethernet pipeline.

Table 3-1. Performance Characterization for the Ethernet Pipeline

Line rate for 4 Gig ports	4 Gigabits/sec
Min Ethernet packet size	64 bytes (+ 20 byte inter packet gap)
Packet Throughput for min packets	5.95 million packets/sec = (4 / (84*8)) * (10**9)
IXP2400 clock frequency	600 MHZ
Inter-packet arrival time for min packets	600/5.95 = 100.84 cycles
Compute cycles per packet for a single microengine	100
Latency per packet for a context pipe single microengine	100 * 8
Compute cycles per packet for n microengines in parallel	100*n
Latency per packet for n microengines in parallel	100*8*n

3.3 Ingress System Resource Allocation

Table 3-2 shows the system resources mapped for the Ingress IXP2400. This mapping reflects the system defaults and may be changed to match the needs of a specific application. The allocation of microengines is done to optimize the performance of this specific application and may be changed for other applications.

Table 3-2. Ingress System Resources Mapped for the Ingress IXP2400

Microblock	ME #	Communication Mechanism with previous stage
Packet RX	ME0	Auto-push status from MSF
IPv4 Forwarder + Ethernet decapsulation/Classify/Filter	ME1, ME2, M5, M6	Scratch ring
Queue Manager	ME3	Scratch ring
CSIX Scheduler	ME4	Next neighbor + Scratch ring
CSIX TX	ME7	Scratch ring

The physical assignment of function to ME is important since it not only affects when the next neighbor registers and signaling can be utilized, but it also affects the utilization of the internal Command bus and S-Push/Pull buses. Since ME0-ME3 belong to Microengine Cluster 0 and ME4-ME7 belong to Microengine Cluster 1, this assignment attempts to balance the usage of the Command bus and S-Push/Pull buses across the two clusters.

IXP2400 supports two SRAM channels and one DRAM channel. Table 3-3 shows the SRAM, DRAM and scratch are utilized for this application. These values are defined in a system header file `dl_system.h` and may be changed as needed.

Table 3-3. SRAM, DRAM, and Scratch Utilization for Ingress System Resource Allocation

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Buffer Descriptors	32	32k (In simulation, we use only 320 buffers)	1 MB		
Buffers	2048	32k		64 MB	
Queue Descriptors	16	1024 (1 per VOQ)	16K		
CSIX TX contexts	32	1024 (1 per VOQ)	32k		
Trie Table	64 (The root Trie table requires at least 257k to support hi64k and hi256 tables. In addition each node requires 64 bytes. These nodes are added as needed)	See note in previous column. Assuming 256k routes, approximately 128k nodes are needed	8MB		
Hash table for MAC Filtering (Ethernet design only)	8	4k	32k		
Route Table (Next Hop Information)	16	Assuming 4k next hops	64k		
IPv4 statistics	4	16			64
Packet RX statistics	4	4*4 (4 per port)			64
IPv4 Directed Broadcast Table	32	256	8k		
Ring from Packet RX to packet processing pipeline (IPv4+Layer2 Decap/ Classify)	20	4k/20			4k
IPv4 to QM ring	12	2k/12			2k
Scheduler to QM	4	512			2k
QM to CSIX TX	8	256			2k
QM Q-Array entries	N/A	16			
Buffer Free list Q-Array entry	N/A	4			

3.4 Egress System Resource Allocation

Table 3-4 shows the system resources allocated for the Egress IXP2400.

Table 3-4. System Resources Allocated for the Egress IXP2400

Microblock	ME #	Communication Mechanism with previous stage
CSIX RX	ME0	Auto-push status from MSF
Packet TX	ME4, ME-5	Scratch ring
Layer-2 Encapsulation	ME3	Scratch ring
Egress QM	ME1	Scratch Ring
Egress Scheduler	ME2	Next neighbor + Scratch ring
Unused (available headroom)	M6, ME7	N/A

The mapping of networking functions onto the microengines shows that six microengines are used to perform the fast path processing for this application. Additional functionality required by customers can be mapped on to the remaining microengines.

Table 3-5 shows how the SRAM, DRAM and scratch are utilized for this application. These values are defined in a system header file `dl_system.h` and may be changed as needed.

Table 3-5. SRAM, DRAM, and Scratch Utilized for Egress System

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM Used	Total Scratch used
Buffer Descriptors	32	32k (In simulation we use only 320 buffers)	1 MB		
Queue Descriptors	16	64 (4 ports x 16 classes per port)	1k		
CSIX RX Reassembly contexts	32	1024 (1 per VOQ)	32k		
Buffers	2048	32k		64 MB	
Layer-2 table with mapping from next hop id to Ethernet header (Ethernet only)	32	Assuming 256 next hops per blade	8k		
CSIX RX to Layer-2 Encap ring	12	512/3 (the size of the ring is 512 long words, but each entry enqueued uses 3 long words. Therefore the total number of entries is 512/3 = 170)			2k
Layer-2 Encap to QM ring	12	512/3			2k
Scheduler to QM ring	4	512			2k
QM to POS TX	4	512512			2k2k
QM Q-Array entries	N/A	15			
Buffer Free list Q-Array entry	N/A	4			

3.5 Interfaces Between the Various Microblocks

The interface between the various microblocks is virtually identical to the POS application as described in Section 2.6, “Interfaces Between the Various Microblocks” on page 34. One difference is the interface between the Queue Manager and the Packet Transmit microblocks on the egress IXP2400.

3.5.1 Packet Queue Manager and Packet TX

The interface between the Packet Queue Manager and the Packet Transmit microengines is four scratch rings—one per Gigabit Ethernet port. Table 3-6 describes each entry in the scratch ring—which is one word.

Table 3-6. One-Word Scratch Ring (Packet Queue Manager and Packet TX)

LW	Bits	Size	Description
0	31:31	1	Valid bit
	30:24	7	Reserved
	23:0	24	Pointer to SOP buffer descriptor in SRAM in long words (Same as bits 0:23 of buffer handle)

3.6 Core Components

The following sections list the core components used by this application.

3.6.1 Ingress Core Components for VxWorks

The core components that run on the Ingress side are Ethernet Rx, IPv4 Forwarder, Stack Driver, Queue Manager, CSIX Tx, and Scheduler. In addition, there are several libraries that are required for the functioning of these core components. These are Route Table Manager, Fragmentation, and Message Support libraries. There is another component called System Application that plays the role of a system designer.

3.6.2 Ingress Core Components for Linux

The core components that run on the Ingress side are Ethernet Rx, IPv4 Forwarder, Queue Manager, CSIX Tx, and Scheduler. In addition, there are several libraries that are required for the functioning of these core components. They are Route Table Manager, Fragmentation, and Message Support libraries. There is another component called System Application that plays the role of a system designer.

3.6.3 Egress Core Components for VxWorks and Linux

The core components that run on the Egress side are Ethernet Tx, Scheduler, CSIX Rx, Queue Manager, and Stack Driver. The libraries that are required on the Egress side are Message Support and L2 Table Manager libraries. There is another component called System Application that plays the role of a system designer.

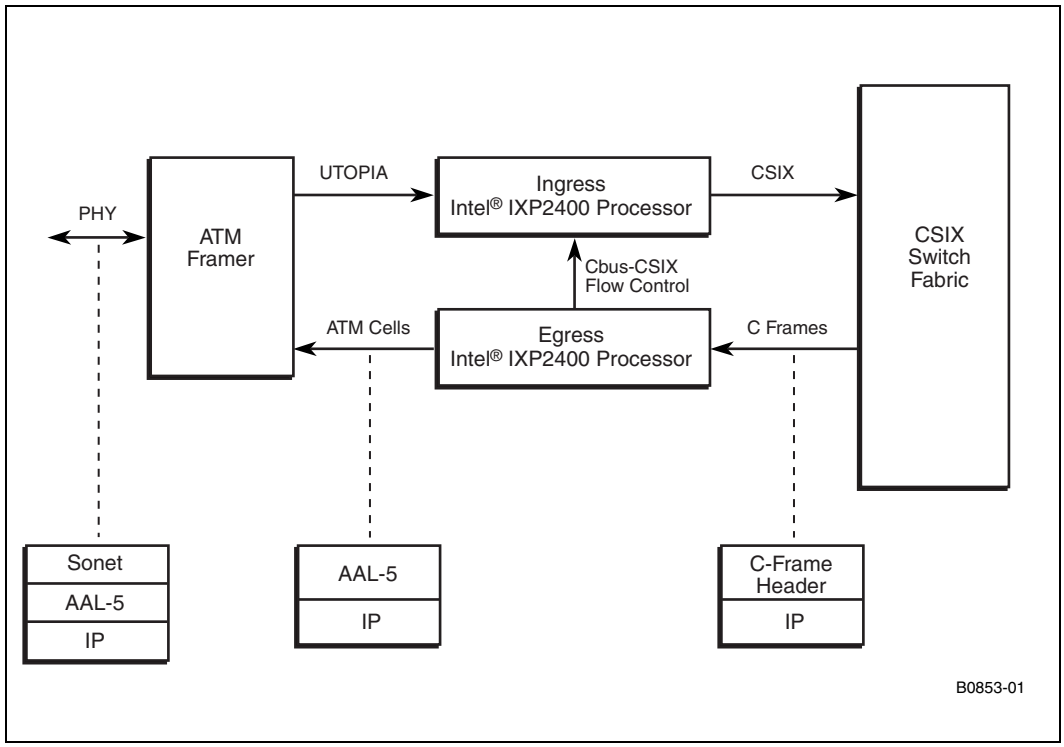
OC-48 ATM IPv4 Forwarding Application 4

This section describes the design of an IPv4 Forwarding application using the IXP2400. Two half-duplex IXP2400's are used to implement an ATM line card at OC-48 data rates that interfaces to a CSIX switch fabric. This section provides a high-level design overview and lists the different software components used to build this application. It focuses only on the fast path or microengine components of the design. The XScale Core Components for this application are described in *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*.

4.1 Hardware Overview for ATM

Figure 4-1 shows two IXP2400's in a typical CSIX full duplex configuration. In this configuration, the two IXP2400's are identified as the ingress processor (receives from the Media interface and transmits to the CSIX Fabric) and the Egress Processor (receives from the CSIX Fabric and transmits to the Media interface).

Figure 4-1. Example Hardware Configuration for OC-48 ATM with CSIX Fabric



The Ingress IXP2400 receives ATM cells. These cells are reassembled into AAL5 frames carrying IP datagrams. The AAL5 header and trailer (along with any LLCSNAP encapsulation) are removed and a Longest Prefix Match (LPM) lookup is performed based on the IPv4 header. The IP

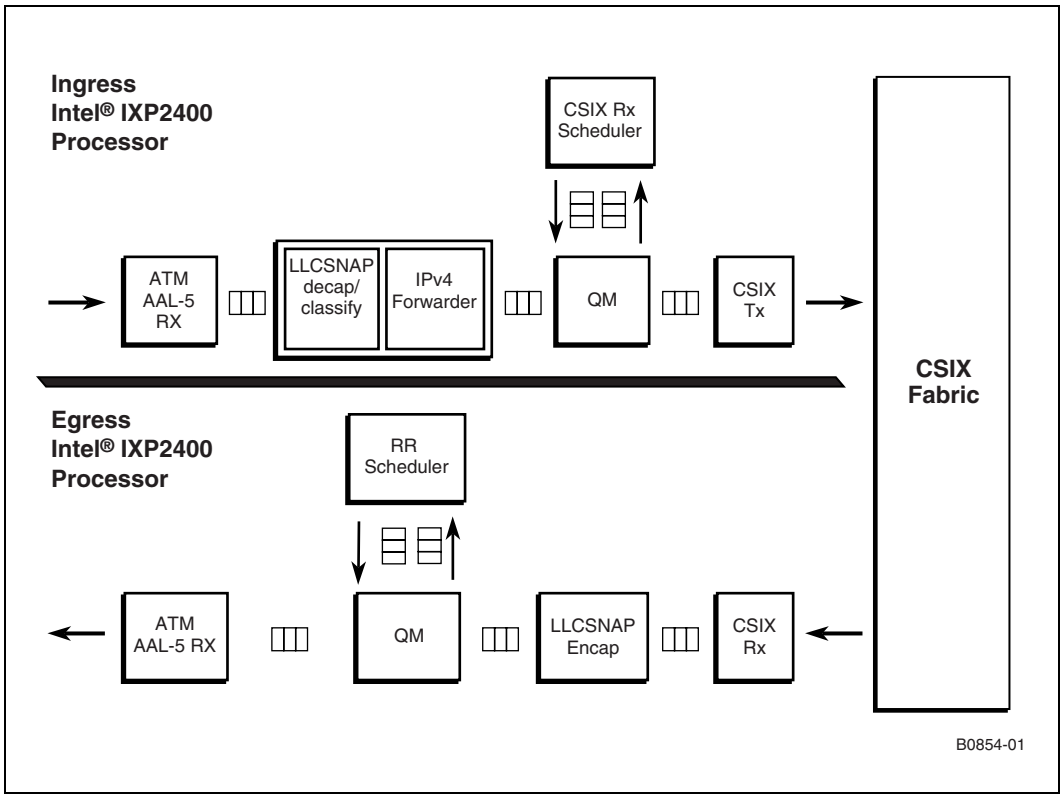
datagrams are then segmented into CSIX C-Frames and transmitted to the CSIX fabric. The result of the LPM lookup determines which IXP2400 connected to the Fabric receives the packet, and which port on that IXP2400 the packet is transmitted on.

The Egress IXP2400 receives CSIX C-Frames from the fabric and reassembles these into IPv4 datagrams. The LLCSNAP headers along with the AAL5 header and trailer information are added to create an AAL5 frame. This AAL5 frame is segmented into ATM cells and transmitted over the appropriate ATM physical port.

4.2 Software Overview for ATM

Figure 4-2 shows the software components needed to implement an IPv4 forwarding application for OC-48 (or 4xOC-12) ATM.

Figure 4-2. Software Components for IPv4 Forwarding Application for OC-48 ATM



4.2.1 Data Flow for the Ingress IXP2400

This section describes the data flow on the Ingress IXP2400:

4.2.1.1 ATM AAL5 RX

The AAL5 Receive microblock on the Ingress IXP2400 receives ATM cells in mpackets coming in on the media interface. It reassembles these cells into AAL5 PDU's, writes the data to a buffer in DRAM and queues the packet buffer handle on a ME-ME scratch ring for processing by the next stage of the pipeline. It also sets up per-packet meta information (offset, size etc) which are passed down the pipeline either in a descriptor in SRAM or in the ME-ME scratch ring itself.

The RX block uses 2 microengines (16 threads) running in parallel to support AAL5 Reassembly at OC-48 data rates. The block supports a compile time option that allows the code to be run such that 4 threads of a microengine work on a single OC-12 port. Up to 64k Virtual Circuits (VCs) are supported and the re-assembly contexts for these are kept in local memory. To maintain packet sequencing and to compute CRC on the incoming cells of a frame, the threads execute in strict order. When the last cell of a frame is received, the block checks if the CRC for the frame is valid. If the CRC is invalid or the length of the packet in the trailer does not match the bytes received for this packet (or the length is 0), the packet is marked to be dropped. Otherwise the packet length in the metadata is adjusted (as per the length field in the AAL5 PDU trailer) to strip the padding and trailer.

Since AAL5 frames may be up to 64k bytes, some large packets may be stored in multiple buffers chained together as a link-list. The buffer handles for the first and last packet in the chain are queued in the scratch ring. Before enqueing the packet buffer handle, the RX block sets up the header field in the packet meta data to be either LLCNSAP or any other protocol indicated by the VC Reassembly Context.

From the AAL5 RX block, the packet is passed on to an application specific system microblock (DL_Sink[]). This microblock checks if the packet has been marked to be dropped (IX_DROP) or sent to the XScale Core (IX_EXCEPTION). If not, it queues the packet buffer handle and associated meta data into the scratch ring for the next stage in the pipeline. OAM cells are queued to the core as exception packets. In addition, AAL5 frames received on any VC may be sent to the XScale Core, if the exception flag is set in the VC Reassembly Context.

4.2.1.2 LLCSNAP Decapsulation and Classify

The LLCSNAP decapsulation/classify microblock runs with the IPv4 microblock on 3 microengines or 24 threads.

This microblock first checks if the header type in the metadata has been set to LLCSNAP. This indicates that the packet is using LLCSNAP encapsulation. If the header is indicated to be LLCSNAP, the microblock removes the header from the packet by updating the offset and size fields in the packet meta data. It also classifies the packet into IPv4, IPv6 etc. If the packet is not using the LLCSNAP encapsulation, the packet classification (`dl_next_block`) is done based on the value of the header type field in the packet metadata. The packet is then sent down the microengine pipeline for further processing. In this application, the dispatch loop silently drops packets classified as IPv6.

4.2.1.3 IPv4 Forwarder

This block is identical to the block described in Section 2.2.1.3, “IPv4 Forwarder” on page 18..

4.2.1.4 Cell Based Queue Manager (Cell QM)

This block is identical to the block described in Section 2.2.1.4, “Cell Based Queue Manager (Cell QM)” on page 18.

4.2.1.5 CSIX Scheduler

This block is identical to the block described in Section 2.2.1.5, “CSIX Scheduler” on page 19.

4.2.1.6 CSIX TX

This block is identical to the block described in Section 2.2.1.6, “CSIX TX” on page 19.

4.2.2 Data Flow for the Egress IXP2400

This section describes the data flow for the Egress IXP2400.

4.2.2.1 CSIX RX

This block is identical to the block described in Section 2.2.2.1, “CSIX RX” on page 20.

4.2.2.2 LLCSNAP Encapsulation

This block adds the LLCSNAP header to the packet and enqueues it to the next stage of the pipeline. It uses the next hop id as an index into a table with layer-2 header information. This table contains both the LLCSNAP header as well as the Virtual Circuit (VC) Queue information for the packet. If the next hop id is set to an invalid value (-1), the block assumes that the layer-2 header has already been added to the packet and sends it to the next stage of the pipeline.

Adding the LLCSNAP header implies that the cell count for the packet needs to be updated. The microblock computes the cell count for the packet

4.2.2.3 Cell Based Queue Manager (Cell QM)

This block is identical to the block described in Section 2.2.1.4, “Cell Based Queue Manager (Cell QM)” on page 18..

4.2.2.4 Round Robin Scheduler

The AAL5 design does not support TM 4.1 Traffic Management. Instead a round robin scheduler is used. For an application that uses ATM TM4.1 refer to the ATM diffserv application.

The round robin scheduler handles two types of queues.

High bit rate queues that transmit traffic up to OC48 rates.

Low bit rate queues that transmit traffic up to OC48/128 rates.

Note that queue 0 is an invalid VCQ.

In OC-48 mode, queue allocation is as shown.

Queues 1-127 are high bit rate (HBR) queues.

Queues 128-65534 are low bit rate (LBR) queues.

In quad OC-12 mode, queue allocation is as shown.

- Queues 1-127 are high bit rate (HBR) queues.
- Queues 128-65534 are low bit rate (LBR) queues.

Port 0:

Queues 1-31 are high bit rate (HBR) queues.
Queues 128, 132, 136 etc are low bit rate (LBR queues).

Port 1:

Queues 32-63 are high bit rate (HBR) queues.
Queue 129, 133, 137 etc are low bit rate (LBR queues).

Port 2:

Queues 64-95 are high bit rate (HBR) queues.
Queue 130, 134, 138 etc are low bit rate (LBR queues).

Port 3:

Queues 96-127 are high bit rate (HBR) queues.
Queue 131, 135, 139 etc are low bit rate (LBR queues).

The algorithm is summarized below

```
1. Read incoming request from NN ring
2. If (enqueue queue number = low bit rate queue)

       If (enqueue transition)

           Schedule a dequeue request for this queue on outgoing scratch ring

   else // (enqueue queue number = high bit rate queue)

       Add cell count to total queue cell count

       Set queue to non-empty
3. If (dequeue queue number = low bit rate queue)

       If (no dequeue transition)

           Schedule a dequeue request for this queue on outgoing scratch ring

   else // (dequeue queue number = high bit rate queue)

       If (dequeue transition)

           Set queue to empty
4. Schedule a dequeue request on a high bit rate queue
```

```
   a. Using round robin select the eligible queue from the set of non-empty
      queues
   b. Schedule a dequeue request for this queue on outgoing scratch ring
   c. Decrement queue cell count for this queue
   d. If queue cell count reaches 0, set the queue to empty
5. Wait for all scratch ring signals and next thread signals
6. Goto step 1
```

4.2.2.5 ATM AAL5 TX

The AAL-5 TX microblock transmits ATM cells over a UTOPIA interface at OC-48 data rates. It receives transmit messages from the queue manager. With each transmit request, the microblock moves an ATM cell into a TBUF, which is then transmitted into the media by the MSF Transmit State Machine.

Every request has an associated AAL-5 frame, which is being segmented into ATM cells. The associated segmentation state for the packet and the packet metadata is maintained in a Transmit Context (TXC) in SRAM. Sixteen TXC's are cached in local memory and the TXC is looked up using the CAM. Like in previous stages, the threads use folding and execute in strict order. If an entire buffer for a packet has been transmitted, then the buffer is freed.

The TX microblock computes the CRC for the AAL-5 frame. For every ATM cell in the frame, the CRC residue (maintained in the TXC) is updated. When the end of the packet is reached, the packet length and CRC are used to prepare an 8-byte AAL-5 trailer, which is also sent out with the remaining payload.

For each request from the QM, the ATM TX microblock processes 48 bytes of the CPCS-SDU. Along with the 48-byte data, it copies a four-byte header with each cell into a TBUF element. When the last cell for a frame is reached, the block processes between 0-48 bytes.

4.2.3 Dispatch Loop

There are two dispatch loops (microblock groups) on the ingress pipeline

- Dispatch Loop for the AAL5 Reassembly Stage (Figure 4-3)
- Dispatch Loop for the IPv4 Forwarder functional pipeline (Figure 4-4)

The QM, Scheduler and CSIX TX blocks don't use a dispatch loop (they still use the dispatch loop macros where required).

Figure 4-3. Dispatch Loop for the AAL5 Reassembly Stage

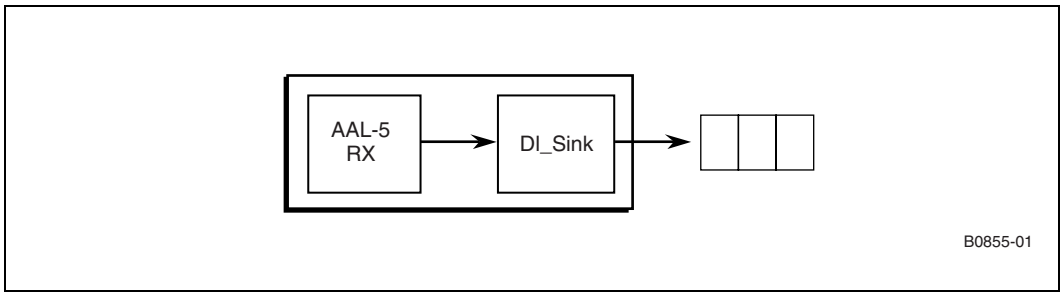
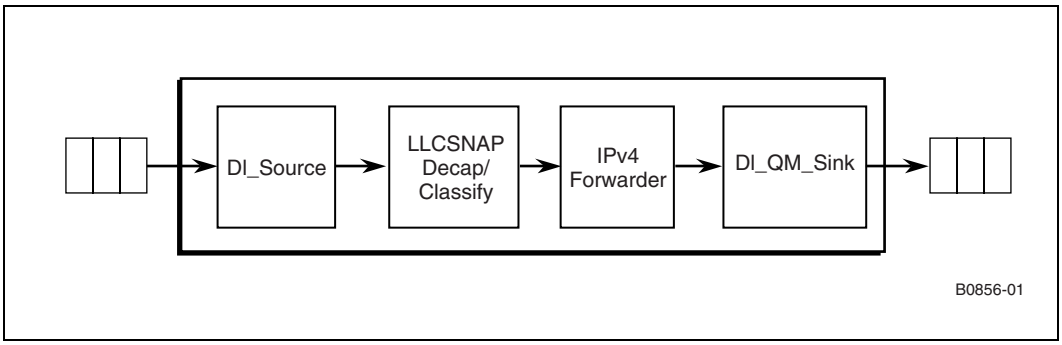


Figure 4-4. Dispatch Loop for the IPv4 Functional Pipeline



Note that the system microblocks dl_source, dl_sink, dl_qm_sink etc are application specific. They may be changed for different packet processing pipelines.

There are two dispatch loops (microblock groups) on the egress pipeline:

- Dispatch Loop for the CSIX RX Reassembly stage (Figure 4-5)
- Dispatch Loop for the LLCSNAP encapsulation stage (Figure 4-6)

Figure 4-5. Dispatch Loop for CSIX Reassembly Stage

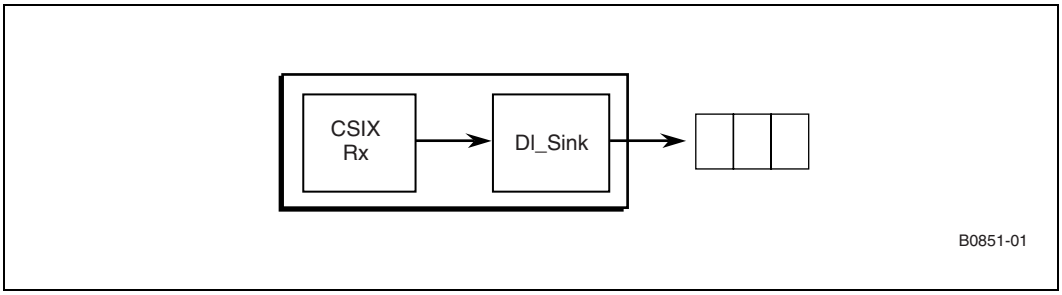
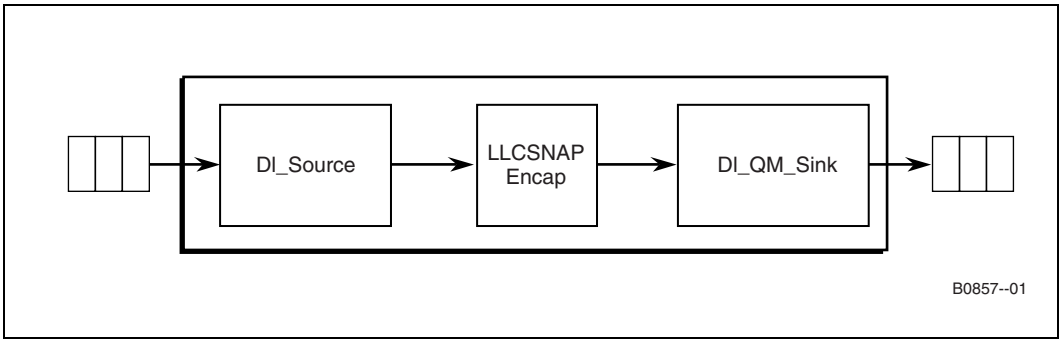


Figure 4-6. Dispatch Loop for LLCSNAP Encapsulation Stage



4.2.4 Performance Characterization

The IXP2400 operates at 600 MHz. For an ATM cell of 53B, the cell inter-arrival time at OC-48 line rate is 105 microengine cycles. The RX/TX blocks need to able to sustain this cell rate. For this design, we assume that the packet has LLCSNAP encapsulation and carries a minimum IP packet in two ATM cells. Therefore the packet inter-arrival time is half of the cell inter-arrival time. This implies that the rest of the pipeline (other than the RX/TX blocks), which process packets and not cells have twice the number of compute cycles per pipe-stage.

Table 4-1 summarizes the performance analysis for the ATM pipeline.

Table 4-1. Performance Characterization for the ATM Pipeline

OC-48 line rate assuming 3% SONET overhead	2.408 Gigabits/sec
ATM cell size	53
Cell Throughput per second	5.67 million cells/sec = (2.408/(53*8)) * (10**9)
Packet Throughput for min packets assuming LLCSNAP encapsulation—2 cells per packet	2.85 million packets/sec = 5.67/2
IXP2400 clock frequency	600 MHZ
Inter-cell arrival time	600/5.67 = 105 cycles
Compute cycles per cell for RX/TX blocks	105 cycles
Latency per cell for RX/TX blocks per microengine	105*8
Inter-packet arrival time for min packets	600/2.85 = 210 cycles
Compute cycles per packet for a context pipe stage	210 cycles
Latency per packet for a context pipe stage	210 * 8
Compute cycles per packet for a functional pipeline of n microengines	210*n
Latency per packet for a functional pipeline of n microengines	210*8*n

4.3 Ingress System Resource Allocation

Table 4-2 shows the system resources mapped for the Ingress IXP2400. This mapping reflects the system defaults and may be changed to match the needs of a specific application. The allocation of microengines is done to optimize the performance of this specific application and may be changed for other applications.

Table 4-2. System Resources Mapped for the Ingress IXP2400

Microblock	ME #	Communication Mechanism with previous stage
AAL-5 RX	ME0, ME1	Auto-push status from MSF
IPv4 Forwarder + LLCSNAP Decapsulation/Classify	ME2, M5, M6	Scratch ring
Queue Manager	ME3	Scratch ring
CSIX Scheduler	ME4	Next neighbor + Scratch ring
CSIX TX	ME7	Scratch ring

Table 4-3 shows the SRAM and DRAM utilized for this application. These values are defined in a system header file dl_system.h and may be changed as needed.

Table 4-3. SRAM and DRAM Utilization for Ingress System Resource Allocation

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Buffer Descriptors	32	32k (In simulation, we use only 320 buffers)	1 MB		
Buffers	2048	32k		64 MB	
Queue Descriptors	16	1024 (1 per VOQ)	16k		
VC Info (RXC context + statistics)	64	64k (1 per VC)	4 MB		
Hash table to find RX context from VPI/VCI/port #	1632	64k 1k	1 MB32 K		
Trie Table	64 (The root Trie table requires at least 257k to support hi64k and hi256 tables. In addition each node requires 64 bytes. These nodes are added as needed)	See note in previous column. Assuming 256k routes, approximately 128k nodes are needed	8MB		
Route Table (Next Hop Information)	16	Assuming 4k next hops	64k		
IPv4 statistics	4	16			64
IPv4 Directed Broadcast Table	32	256	8k		
Ring from RX to packet processing (IPv4+Layer2 Decap/ Classify)	16	256			4k
IPv4 to QM ring	12	512/3 (the size of the ring is 512 long words, but each entry enqueued uses 3 long words. Therefore the total number of entries is 512/3 = 170)			2k
Scheduler to QM	4	128			512
QM to CSIX TX	8	256			512
QM Q-Array entries	N/A	16			
Buffer Free list Q-Array entries	N/A	4			

4.4 Egress System Resource Allocation

Table 4-4 shows the system resources allocated for the Egress IXP2400.

Table 4-4. System Resources Allocated for the Egress IXP2400

Microblock	ME #	Communication Mechanism with previous stage
CSIX RX	ME0	Auto-push status from MSF
ATM TX[1]*	ME5, ME6, ME7	Scratch ring
Layer-2 Encapsulation	ME1	Scratch ring
Cell QM	ME2	Scratch Ring
Round robin scheduler	ME3	Next Neighbor

1. OC-48 configuration uses 3 ME's. For quad OC-12, only 2 ME's (ME5, ME6) are used.

Table 4-5 shows the SRAM and DRAM utilized for this application. These values are defined in a system header file `dl_system.h` and may be changed as needed.

Table 4-5. SRAM and DRAM Utilization for Egress System Resource Allocation

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Buffer Descriptors	32	32k (In simulation we use only 320 buffers)	1 MB		
Queue Descriptors	16	64k (1 per VC)	1 MB		
CSIX RX contexts	64	1024	64K		
Buffers	2048	32k		64 MB	
Layer-2 table with mapping from next hop id to VPI/VCI and VCQ	16	4k (1 per next hop)	64k		
CSIX RX to Layer-2 Encap ring	12	512/3 (the size of the ring is 512 long words, but each entry enqueued uses 3 long words. Therefore the total number of entries is 512/3 = 170)			2k
Layer-2 Encap to QM ring	12	512/3			2k
Scheduler to QM ring	4	512			2k
QM to ATM TX	8	256			256
QM Q-Array entries	N/A	15	15		15
Buffer Free list Q-Array entries	N/A	4	4		4

4.5 Interfaces Between the Various Microblocks

This section describes the interfaces between the different microblocks on the ingress and egress processors for this application.

In most of the messages, there is a valid bit is used to prevent a value of zero from being enqueued on the scratch ring. Zero is used to detect a case where the scratch ring is empty. So the valid bit helps distinguish between a zero value that was actually enqueued versus a case where the ring is empty.

4.5.1 AAL5 RX and Packet Processing Microengines

The interface between the AAL5 RX Microblock and the Packet Processing Microengines (IPv4+L2 decap) running the layer-2 decapsulation/classify and IPv4 forwarding code is a scratch ring. Table 4-6 describes each entry in the scratch ring—which is six words.

Table 4-6. Six-Word Scratch Ring Entry (IPv4+L2 Decap)

LW	Bits	Size	Field	Description
0	31:0	32	dl_buffer_handle	Buffer Handle for the SOP Descriptor
1	32:0	32	dl_eop_buffer_handle	Buffer Handle for the EOP Descriptor
2	31:16	16	buffer_size	Buffer size in bytes
	15:0	16	offset	Offset of the start of data in the buffer in bytes
3	31:28	16	packet_size	Total packet size across buffers
	15:12	4	free_list_id	Free list ID for buffer
	11:8	4	rx_stat	Receive Status Flag
	7:0	8	header_type	Type of header at offset bytes into the packet
4	31:16	16	input_port	Input port on ingress processor
4	15:0	16	reserved	Reserved
5	32:0	32	flow_id	VC key (8-bit VPI, 16-bit VCI and 4-bit input port)

4.5.2 Packet Processing Microengines and Cell Queue Manager

This interface is identical to the POS application described in Section 2.6.2, “Packet Processing Microengines and Cell Queue Manager” on page 27.

4.5.3 Cell Queue Manager and CSIX Scheduler

This interface is identical to the POS application described in Section 2.6.3, “Cell Queue Manager and CSIX Scheduler” on page 28.

4.5.4 Cell Queue Manager and CSIX TX

This interface is identical to the POS application described in Section 2.6.4, “Cell Queue Manager and CSIX TX” on page 28.

4.5.5 CSIX RX and LLCSNAP Encapsulation

The CSIX RX and LLCSNAP Encapsulation interface is a scratch ring. Table 4-7 describes each entry in the scratch ring—which is three words.

Table 4-7. Three-Word Scratch Ring (CSIX RX and LLCSNAP Encap)

LW	Bits	Size	Field	Description
0	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
1	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)
2	31	1	Valid Bit	Must be 1
2	30:18	13	Reserved	Reserved
2	17:16	2	Port number	Port Number
2	0:15	16	Queue Number	Queue Number

4.5.6 LLCSNAP Encap and Cell Queue Manager

The interface between the LLCSNAP Encap microblock and the Cell Queue Manager is a scratch ring. Table 4-8 describes each entry in the scratch ring—which is three long words.

Table 4-8. Three-Word Scratch Ring (LLCSNAP Encap and Cell QM)

LW	Bits	Size	Field	Description
0	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
1	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)
2	31	1	Valid Bit	Must be 1
2	30:29	2	Reserved	Reserved
2	28:18	11	Packet cell count	Number of 48-byte cells in the entire packet
2	17:16	2	Output port number	Output port number
2	15:0	16	Queue Number	Queue Number

4.5.7 Cell Queue Manager and RR Scheduler for ATM

This is similar to the interface for POS and Ethernet except that the cell count for the packet is sent to the shaper block on each enqueue, while the packet length is not required for the dequeue. Therefore for each enqueue request to the Queue Manager, a message is sent to the scheduler block. For dequeue requests, only transitions are sent to the scheduler. In any iteration, if there is no enqueue request and a dequeue transition occurs, the valid bit is set to zero in the first word of the message.Table 4-9 shows the Cell Queue Manager and scheduler for ATM.

.

Table 4-9. Cell Queue Manager and RR Scheduler for ATM

LW	Bits	Size	Field	Description
0	31	1	Valid Bit	The enqueue word is valid only if this bit is set
	30	1	Enqueue Transition	Notification that queue has gone from empty to non-empty
	29:16	1	Reserved	Reserved
	28:18	11	Packet cell count	Unused for POS/Ethernet
	17:16	2	Output port number	Output port number
	15:0	16	Queue Number	Queue Number that was enqueued
1	31	1	Valid Bit	Must be 1
	30	1	Dequeue Transition	Notification that queue has gone from non-empty to empty
	29	1	Invalid Dequeue	Unused for ATM
	28:16	13	Packet size	Unused for ATM
	15:0	16	Queue Number	Queue Number that was dequeued

4.5.8 RR Scheduler to Cell Queue Manager

The interface between the RR Scheduler and the Cell Queue Manager is a scratch ring. Table 4-10 describes each entry in the scratch ring—which is one long word.:

Table 4-10. One-Word Scratch Ring Entry (TM 4.1 Scheduler to Cell Queue Manager)

LW	Bits	Size	Field	Description
0	31	1	Valid Bit	Must be 1
0	30:18	13	Reserved	Reserved
0	16:17	2	Output port number	Output port number
0	0:15	16	Queue Number	Queue Number

4.5.9 Cell Queue Manager and AAL-5 TX

The interface between the Cell Queue Manager and AAL-5 TX microblock is a scratch ring. Table 4-11 describes each entry in the scratch ring—which is two long words.:

Table 4-11. Two-Word Scratch Ring Entry (Cell Queue Manager and AAL-5 TX)

LW	Bits	Size	Field	Description
0	31	1	Valid Bit	Must be 1
0	30:30	1	Reserved	Reserved
0	29:19	11	Output port number	Output port number

Table 4-11. Two-Word Scratch Ring Entry (Cell Queue Manager and AAL-5 TX)

LW	Bits	Size	Field	Description
0	18:16	3	Reserved	Reserved
0	15:0	16	Queue Number	Queue Number
1	31:0	32	Buffer Handle	Buffer Handle currently being transmitted for queue

OC-192 POS IPv4/IPv6 Forwarding/ Tunneling Application 5

This section describes the design of an IPv4/IPv6 forwarding and tunneling application using the Intel® IXP2800 Network Processor. Two half-duplex IXP2800 processors are used to implement a POS line card at OC-192 data rates that interfaces to a CSIX switch fabric. This section provides a high-level design overview and lists the different software components used to build this application. It focuses only on the fast path or microengine components of the design. The Intel XScale® core components for this application are described in the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*.

The application described in this chapter is supported on the Intel® IXDP2800 Advanced Development Platform.

Note: This application has been ported from the OC-48 POS IPv4 Forwarding application for the Intel® IXP2400 Network Processor. This section describes in detail the differences between the IXP2400 application and the IXP2800 application and the methodology used to port from one processor to the other.

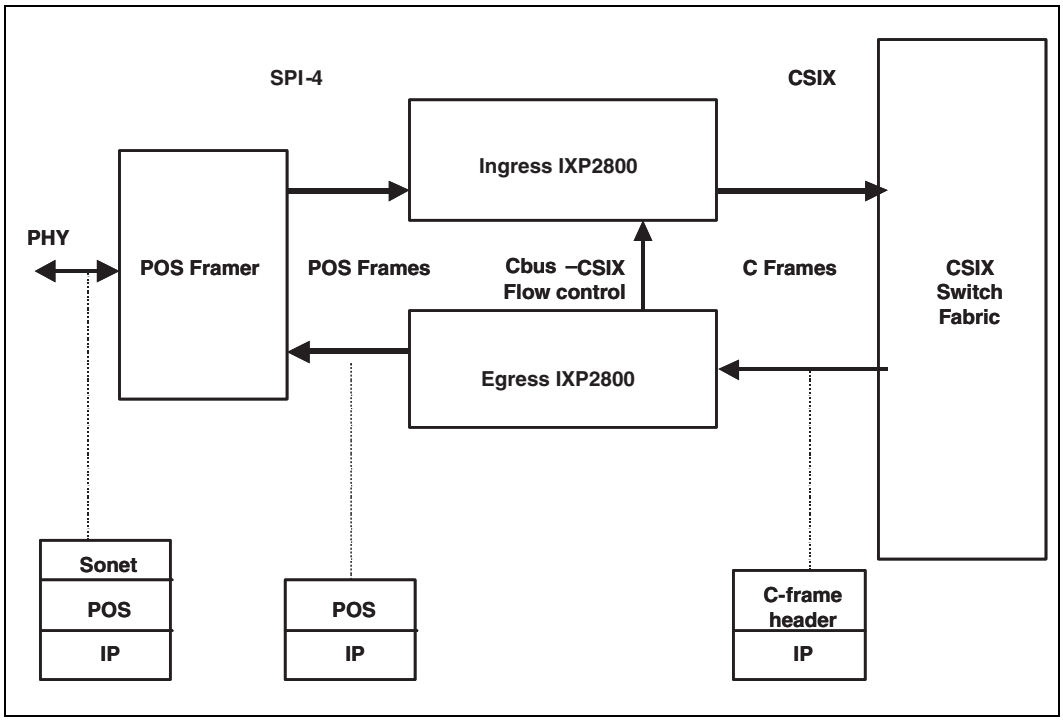
5.1 Hardware Overview

Figure 5-1 illustrates an example hardware configuration for OC-192 POS line card with CSIX fabric. The figure shows two IXP2800 processors in a typical CSIX full duplex configuration. In this configuration, the two IXP2800 processors are identified as the ingress processor (receives from the Media interface and transmits to the CSIX Fabric) and the egress processor (receives from the CSIX Fabric and transmits to the Media interface).

The Ingress IXP2800 receives POS frames that carry IPv4 datagrams. The frames are assembled into IPv4 or IPv6 packets and the Layer-2 (PPP) headers are removed. Based on the IPv4 or IPv6 header, a Longest Prefix Match (LPM) lookup is performed and the packets are segmented into CSIX C-Frames and transmitted to the CSIX fabric. The result of the LPM lookup determines which IXP2800 connected to the Fabric receives the packet, and which port on that IXP2800 the packet is transmitted on.

The Egress IXP2800 receives CSIX C-Frames from the fabric and reassembles these into IPv4 or IPv6 datagrams. The Layer-2 (PPP) headers are added and the packets are transmitted over the appropriate port.

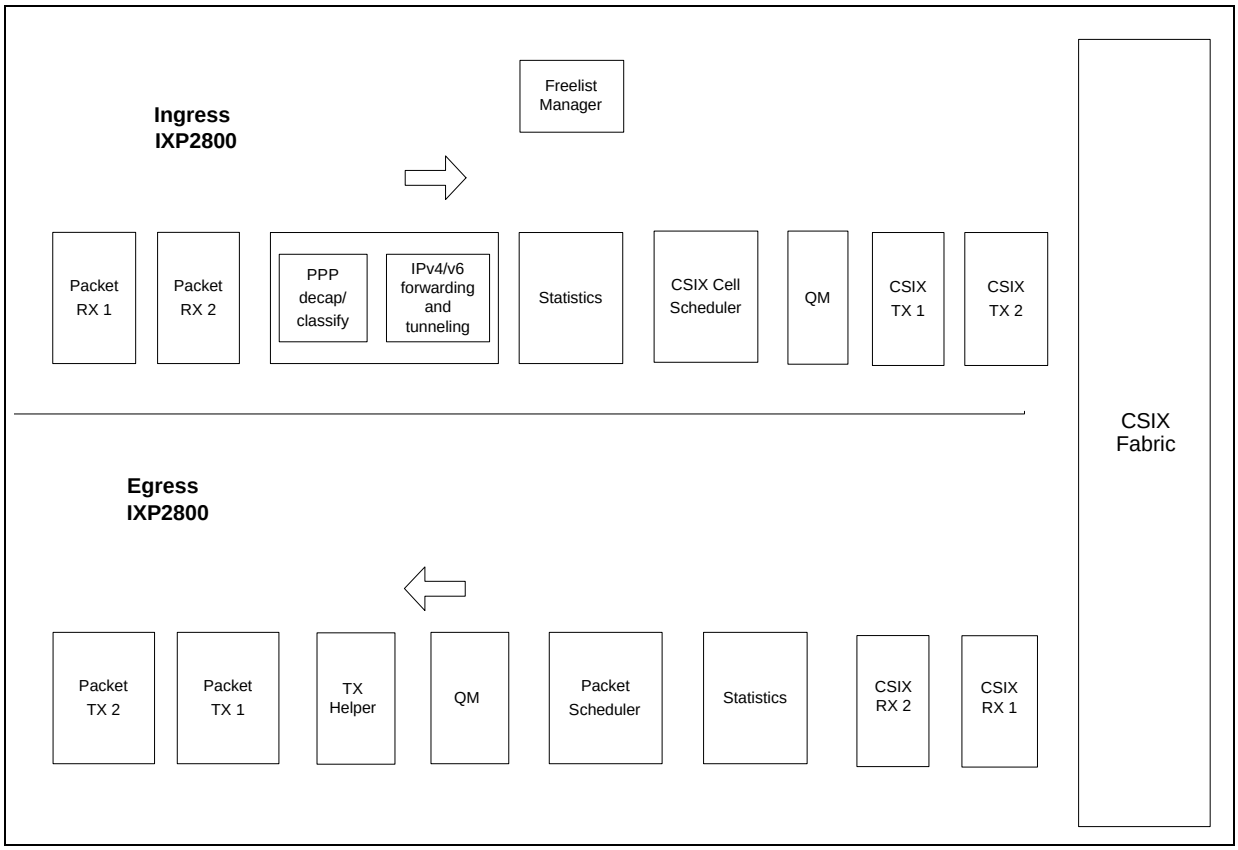
Figure 5-1. Example Hardware Configuration for OC-192 POS Line Card with CSIX Fabric



5.2 Software Overview

Figure 5-2 illustrates the microblocks needed to implement an OC-192 POS IPv4/IPv6 forwarding and tunneling application. The design for this application is based on the guidelines specified by the IXA Portability Framework—*Intel® Internet Exchange Architecture Portability Framework Developer's Manual*. The driver microblocks (Receive, Transmit, Scheduler and QM) run on different microengines from the packet processing code.

Figure 5-2. Microblocks for an OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application



5.2.1 Data Flow for the Ingress IXP2800

The data flow is essentially the same as the OC-48 POS IPv4/v6/tunneling Forwarding application described in Chapter 2, “OC-48 POS IPv4 Forwarding Application”. This section highlights the differences between the two applications.

5.2.1.1 Packet RX

The Packet RX microblock runs on two microengines in a context pipeline connected by a Next Neighbor ring. The Packet RX microblock for the IXP2400 (Section 2.2.1.1, “Packet RX” on page 25) has been extended such that as a compile time option it now runs on two microengines.

This microblock performs frame-reassembly on the mpackets coming in on the POS media interface. It reassembles and writes the packet data to a buffer in DRAM and queues the packet buffer handle on a ME-ME scratch ring for processing by the packet processing microengines. It also sets up per- packet meta information (offset, size etc) which are passed on either in a descriptor in SRAM or in the ME-ME scratch ring itself. Up to 16 virtual ports are supported and the re-assembly context for all these ports is kept in local memory. To maintain packet sequencing, the threads execute in strict order. The microblock is written such that it supports up to 16 virtual ports, but one or more of these may be unused. This allows the same microblock to support different configurations such as Quad-OC48, 16 OC-12, or a single OC-192 port.

In this application, the packets reassembled are PPP frames containing IP datagrams. RFC 2615 defines the Packet Over SONET specification and refers to RFC 1661 (PPP) and RFC 1662 (PPP in HDLC-like framing). PPP framing including header validation, FCS generation and computation and byte stuffing are handled by the POS framer (IXF 18101).

Since POS packets may be up to 9k bytes, some large packets may be stored in multiple buffers chained together as a link-list. The buffer handles for the first and last packet in the chain are queued in the scratch ring.

From the Packet RX block, the packet is passed on to an application specific system microblock (`DL_Sink[]`). This microblock checks if the packet has been marked to be dropped (`IX_DROP`) or sent to the XScale Core (`IX_EXCEPTION`). If not, it queues the packet buffer handle and associated packet meta data into the scratch ring for the next stage in the pipeline.

5.2.1.2 Packet Processing Microengines (PPP Decap/Classify + IPv4/IPv6 Forwarder/Tunneling)

The PPP decapsulation/classify microblock runs along with the IPv4/IPv6 forwarding/tunneling microblocks. These microblocks are identical to the ones used for the IXP2400 POS application described in Section 2.2.1.2, “PPP Decapsulation and Classify” on page 25, Section 2.2.1.3, “IPv4 Forwarder” on page 26, and Section 7.2.6, “IPv6/IPv4 Tunneling Microblock” on page 97.

An application specific system source microblock on each thread dequeues packet buffer handles from the scratch ring. This source block (`DL_Source[]`) is a system microblock implicit in the dispatch loop. It reads in the packet meta information—that is, the packet descriptor, and populates the dispatch loop state. It also reads in up to 40 bytes of the packet header from DRAM into a header cache maintained in transfer registers. Since it is important to maintain packet sequencing, the threads in the microblock execute in strict order to dequeue from the scratch ring. This implies that the first thread on microengine 1 dequeues the first packet, signals the next thread to perform the dequeue and so on. From this block, the packet goes to the PPP decapsulation/classify microblock.

The PPP decapsulation/classify microblock removes the layer-2 PPP header from the packet by updating the offset and size fields in the packet descriptor. Based on the PPP header, it also classifies the packet into IPv4, IPv6, PPP control packet (LCP, IPCP). If the packet is a PPP control packet, it is marked as an exception packet to be sent to the Intel XScale® core (`IX_EXCEPTION`). Otherwise the packet is sent down the microengine pipeline for further processing.

The IPv4 forwarder microblock validates the IP header per RFC 1812. If the validity checks fail, then the packet is set up to be dropped as specified in *Intel® Internet Exchange Architecture Portability Framework Developer’s Manual*. Otherwise a Longest Prefix Match (LPM) is performed on the IPv4 header. The result is an IPv4 Next Hop ID, a fabric blade id (identifying a unique IXP2800 on the fabric) and an output port identifying the output port on the egress IXP2800. The Next Hop ID is passed over the CSIX fabric to an Egress IXP2800 where it is used to look up information about the Layer-2 header to be prepended to the packet buffer. The output port is also passed over the CSIX fabric to the egress IXP2800 and is used to transmit over the appropriate port. All three fields are stored in the packet meta data—that is, the packet descriptor.

If the packet is an IPv6 packet encapsulated into an IPv4 packet, the IPv4 forwarder sends the packet to the tunneling decap microblock. After removing the IPv4 header, the tunneling decap microblock sends the packet to the IPv6 forwarder for IPv6 forwarding.

The IPv6 forwarder microblock processes an IPv6 packet in a manner similar to the IPv4 forwarder.

If the packet needs to be encapsulated into an IPv4 packet, the IPv6 forwarder sends the packet to the tunneling encap microblock. After inserting an IPv4 header, the tunneling encap microblock sends the packet to the IPv4 forwarder for IPv4 forwarding.

If no match is found, then the packet is set up to be sent up to the Intel XScale® core for further processing as specified in the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*. Packets are also sent to the core in a number of other cases, for example, when the packet is destined for a local interface or is to be fragmented.

Finally, the packet is passed on to an application specific system microblock (`DL_QM_Sink[]`). This microblock checks if the packet is to be dropped or sent to the Intel XScale® core. If not, it sends an enqueue request to the Statistics microengine over a scratch ring. The `DL_QM_Sink[]` microblock also writes the cached packet header to DRAM and the packet meta information to SRAM.

5.2.1.3 Statistics Microblock

This microblock runs on a single microengine. It is currently a place holder for statistics handling. It is anticipated that when this application is extended for MPLS and DiffServ, this microblock is used to manage per-flow statistics.

Note: The design for handling statistics will be described in future revisions of the document.

The statistics microengine interfaces to the IXP2800 CSIX Fabric Scheduler microblock via a Next Neighbor ring passing it the packet enqueue requests received from the packet processing microengines. It also computes the total cell count of every packet enqueued and passes it to the scheduler. In addition, it also handles dropping of large packets that are stored in multiple buffers.

5.2.1.4 CSIX Scheduler

The CSIX scheduler runs on a single microengine and schedules c-frames into the CSIX fabric. This microblock is significantly different from the one currently used on the IXP2400. It has been optimized to run in 57 cycles which is the min POS packet instruction budget. Also it is placed in the packet processing pipeline before the queue manager allowing it to keep track of enqueue and dequeue transitions correctly and without any latency. Unlike the IXP2400 version which handles 1024 VOQs, the design used for the IXP2800 supports 256 VoQs.

The scheduling algorithm implemented is Round Robin among the ports on the fabric and Weighted Round Robin among the queues on a port. Since this is not a QoS application and there is only one queue per port, the Weighted Round Robin scheduling degenerates to round robin scheduling. Other applications, e.g. IP DiffServ may use the WRR functionality. The scheduling and transmit is done a cframe at a time.

The CSIX scheduler handles the following:

- Flow control messages from the fabric. These messages are sent by the fabric to the egress IXP2800, which sends them on the c-bus to the ingress IXP2800. If the fabric asserts Xoff on a particular VoQ (Virtual Output Queue), the scheduler stops scheduling for the queue.
- Packet enqueue requests from the previous microengine. It uses this information to update a list of active queues (queues with data) and to track queue transitions (empty to non-empty and vice-versa). A queue is scheduled only if there is data in the queue. The enqueue requests are passed on via Next Neighbor ring to the Queue Manager.

- MSF Transmit State Machine. The scheduler monitors how many packet cframes are in the pipeline and if it exceeds a certain threshold, it stops scheduling.

During each loop, the scheduler also

- Checks its list of active queues (queues with data). Picking up from where it left off in the last iteration it finds the next queue to schedule.
- It then sends a dequeue message to the Queue Manager to dequeue the head of that queue. The Queue Manager dequeues a cell (cframe) from the head of the queue and sends a transmit request to the CSIX TX microblock.

5.2.1.5 Cell Based Queue Manager (Cell QM)

The Queue Manager (QM) is a driver microblock that runs on a single microengine. This microblock is significantly different from the one currently used in the IXP2400 application. It has been optimized to run within 57 cycles which is the instruction budget for a min POS packet at OC-192 data rates. The key difference is that in the IXP2800 design, the scheduler keeps track of the queue size and queue transitions. This considerably simplifies the Queue Manager which no longer has to support this functionality.

The QM manages enqueue and dequeue operations on the transmit queues which are implemented using the hardware SRAM link lists. It accepts enqueue requests from the scheduler via a Next Neighbor ring. The enqueue requests are on a per-packet basis. The dequeue requests are on a per-cell basis where a cell is a CSIX cframe.

The threads on the QM microengine execute in strict order using local inter-thread signaling. SRAM Queue Array entries are cached in the SRAM controller and the CAM is used for managing the tags for these. To maintain coherence among threads, folding is used.

5.2.1.6 CSIX TX

The CSIX Transmit microblock runs on two microengines in a context pipeline connected by a Next Neighbor ring. The CSIX Transmit microblock for the IXP2400 (Section 2.2.1.6, “CSIX TX” on page 27) has been extended so that as a compile time option it now runs on two microengines.

This microblock receives transmit messages from the queue manager via a Next Neighbor ring. With each transmit request, the microblock moves a cframe into a TBUF, which is then transmitted into the fabric by the MSF Transmit State Machine.

Every request has an associated packet, which is being segmented into cframes. The associated segmentation state for the packet and the packet metadata is cached in local memory and is looked up using the CAM. The TX microblock adds the CSIX header onto the cframe along with the packet data. Along with the CSIX header, a Traffic Manager (TM) header is also added per cframe carrying extra information (destination Layer-2 port id, input blade id, sequence number, next-hop id etc.) about the packet to be passed to the Egress IXP2800. In addition, the flow id, class id, input port and some other fields from the metadata are passed along to the Egress IXP2800 using a per-packet header pre-pended to the start of the first c-frame of each packet.

5.2.1.7 Freelist Manager

This microblock maintains the packet buffer freelist. It replaces the linked list scheme that uses SRAM and the Q-array hardware for maintaining a packet buffer freelist.

Full details of the Freelist Manager microblock are contained in the *Intel® Internet Exchange Architecture Software Building Blocks Developer's Manual*, Chapter 38, "Freelist Manager".

5.2.2 Data Flow for the Egress IXP2800

This section describes the data flow for the Egress Intel® IXP2800 Network Processor.

5.2.2.1 CSIX RX

The CSIX Receive microblock executes on two microengines or 16 threads. The CSIX Receive microblock for the IXP2400 (Section 2.2.2.1, "CSIX RX" on page 28) has been extended such that as a compile time option it now runs on two microengines.

This microblock receives c-frames from a CSIX fabric and reassembles them into IP packets. Since the packets being reassembled may be up to 9k bytes, some large packets may be stored in multiple buffers chained together as a link-list. The buffer handles for the first and last packet in the chain are queued in the scratch ring. The CSIX RX microblock also sets up packet meta information (offset, size, and so on) which are passed on to the next microengine either by writing to the SRAM packet descriptor or via the Next Neighbor ring.

5.2.2.2 Egress Packet Scheduler

The Egress scheduler schedules POS packets to be transmitted over the POS interface. This is a packet-based scheduler as opposed to the cell—that is, c-frame based scheduler on the Ingress side.

The scheduler is a Deficit Round Robin scheduler, as described in the *Intel® IXA Building Blocks: Developer's Manual*, Chapter 20, "OC-192 DRR Egress Scheduler". The packet scheduler is a context pipe-stage that is implemented as a microblock that runs on 3 microengines. This microblock includes the Class Schedule block, the Count block, and the Port Schedule block. Each block runs on one microengine.

The packet scheduler supports up to 16 virtual ports. Since these ports may have differing bandwidth requirements, the scheduler implements Weighted Round Robin (WRR) scheduling on the ports. This allows us to support different configurations (16 OC-3, 4 OC-12, 1 OC-48 etc) simply by adjusting the weights for the ports in the scheduler.

For each port, the scheduler supports up to256 queues per port. The Scheduler implements a modified version of Deficit Round Robin (DRR) scheduling on the queues within a port.

Since there is no QoS requirement in the application, we will only use one of the classes per port. This means there is only one queue per port and the DRR scheduling is unused in application. However the same code can be reused in a QoS Diffserv application in which case the DRR scheduling is applicable.

The scheduler also keeps track of the number of packets in flight (scheduled, but not transmitted) for each port. If this number exceeds a specified limit, then it stops scheduling on that port.

Note: This scheduler is currently fully tested only in simulation mode. In a future release it will be tested on hardware. Currently we use a simple round robin scheduler when running this application on hardware.

5.2.2.3 Packet Based Queue Manager (Packet QM)

This block is almost identical to the Ingress Queue Manager except that it dequeues packets. The SRAM Q-Array hardware is programmed in packet mode and ignores the cell count field in the buffer handle.

5.2.2.4 TX Helper

This block acts a helper to the Packet Tx and the Packet Scheduler microblock

It gets TX requests from the Packet QM block via the Next Neighbor ring. For multi-port applications, it sends the request to the appropriate scratch ring that is read by Packet Tx. For single port applications such as this one, this is not required.

It updates the per-class counters in SRAM. These counters keep tracks of the number of packets transmitted per class for the DRR Packet Scheduler. To do this, the Tx Helper block reads packet meta data to find the class ID for each packet. Then it calculates the SRAM address of the counter, reads the counter, increments the content, and writes back the new value.

5.2.2.5 Packet TX

The Packet Transmit microblock transmits packets over the POS media interface. It runs on two microengines in a context pipeline connected by a Next Neighbor ring. It segments a packet into mpackets, and moves them into TBUFS for the MSF state machine to transmit. The Packet TX microblock supports a single OC-192 POS port.

The Packet TX microblock monitors the MSF to see if the TBUF threshold for a specific port has been exceeded. If so it stops transmitting on that port and any requests to transmit packets on that port are queued up in local memory. This microblock also periodically updates the scheduler with information about how many packets have been transmitted. If the packets in flight for a particular port (packets scheduled but not transmitted) exceed a certain limit (which depends on the bandwidth supported by that port), then the scheduler stops scheduling any more packets for the port.

5.3 Performance Characterization

The Intel® IXP2800 Network Processor operates at 1400 MHz. For a min POS packet of 49B, the packet inter-arrival time at OC-192 line rate is 57 microengine cycles. In order to maintain line rate for min packets, each stage of the pipeline cannot exceed this budget. In other words, each stage of the pipeline needs to retire a packet every 57 cycles.

Table 5-1 summarizes the performance analysis for the POS pipeline.

Table 5-1. Performance Analysis for the POS Pipeline

OC-192c line rate assuming 3% SONET overhead	9.62 Gigabits/sec
Min POS packet size	49 bytes (40 byte TCP/IP, 2 bytes Address and Control, 2 byte PPP header, 4 byte FCS and 1 byte flag)
Packet Throughput for min packets	24.56 million packets/sec = (9.62 / (49*8)) * (10**9)
IXP2400 clock frequency	1400 MHz

Table 5-1. Performance Analysis for the POS Pipeline

OC-192c line rate assuming 3% SONET overhead	9.62 Gigabits/sec
Inter-packet arrival time for min packets	1400/6.14 = 57 cycles
Compute cycles per packet for a single microengine	57
Latency per packet for a single microengine	57 * 8
Compute cycles per packet for n microengines running in parallel	57*n
Latency per packet for n microengines running in parallel	57*8*n

5.4 Ingress System Resource Allocation

Table 5-2 shows the system resources mapped for the Ingress IXP2800. This mapping reflects the system defaults and may be changed. The allocation of microengines is done such that it optimizes the performance of this specific application and may be changed for other applications.

Table 5-2. System Resources Mapped for the Ingress IXP2800

Microblock	ME #	Communication Mechanism with previous stage
Packet RX	ME 1:3, 1:4	Auto-push status from MSF
IPv4 Forwarder + Layer2 decapsulation/Classify	ME 0:0, 0:1, 0:2, 0:3, 0:4, 1:5, 1:6, 1:7	Scratch ring
Statistics	ME 0:5	Scratch ring
CSIX Scheduler	ME 0:6	NN ring
Queue Manager	ME 0:7	NN ring
CSIX TX	ME 1:0, 1:1	NN ring
Freelist Manager	ME 1:2	NN ring

The physical assignment of function to microengine is important since it not only affects when the next neighbor registers and signaling can be utilized, but it also affects the utilization of the internal Command bus and S-Push/Pull buses. This assignment attempts to balance the usage of the Command bus and S-Push/Pull buses across the two clusters.

The IXP2800 supports four SRAM channels and three DRAM channel. Table 5-3 shows the SRAM, DRAM and scratch utilized for this application. These values are defined in a system header file `dl_system.h` and may be changed as needed.

Table 5-3. SRAM, DRAM, and Scratch Utilization for Ingress IXP2800

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Buffer Descriptors	32	32k (In simulation, we use only 320 buffers)	1 MB		
Buffers	2048	32k		64 MB	
Queue Descriptors	16	256 (1 per VOQ)	4K		

Table 5-3. SRAM, DRAM, and Scratch Utilization for Ingress IXP2800 (Continued)

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
CSIX TX contexts	32	256 (1 per VOQ)	8k		
Trie Table	64 (The root Trie table requires at least 257k to support hi64k and hi256 tables. In addition each node requires 64 bytes. These nodes are added as needed)	See note in previous column. Assuming 256k routes, approximately 128k nodes are needed	8MB		
Route Table (Next Hop Information)	16	Assuming 4k next hops	64k		
Tunnel Encap (Next Hop Information)	32	256	8KB		
Tunnel Decap(Next Hop Information)	4	128	512B		
V6V4 Ingress Source List	64	256	16KB		
IPv4 statistics	4	16			64
Packet RX statistics	4	16*16	1024		
IPv4 Directed Broadcast Table	32 (local memory)	64			
Ring from Packet RX to packet processing pipeline (IPv4+Layer2 Decap/Classify)	12	4k/3			4k
IPv4 to Statistics ring	12	2k/12			2k
QM Q-Array entries	N/A	16			
Buffer Free list Q-Array entry	N/A	4			

5.5 Egress System Resource Allocation

Table 5-4 shows the system resources allocated for the Egress IXP2800.

Table 5-4. System Resources Allocated for Egress IXP2800

Microblock	ME #	Communication Mechanism with previous stage
CSIX RX	ME 1:1, 1:2	Auto-push status from MSF
Statistics	ME 0:0	Scratch ring
DRR Scheduler	ME 0:1, 0:2, 0:3	NN ring
Queue Manager	ME 0:4	NN ring
TX Helper	ME 0:5	NN ring
Packet TX	ME 0:6, 0:7, 1:0	NN ring

The mapping of networking functions on to the microengines shows that 9 microengines are used to perform the fast path processing for this application. Additional functionality required by customers can be mapped on to the remaining microengines.

Table 5-5 shows the SRAM, DRAM and scratch utilized for this application. These values are defined in a system header file `dl_system.h` and may be changed as needed.

Table 5-5. SRAM, DRAM, and Scratch Utilization for Egress IXP2800

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM Used	Total Scratch used
Buffer Descriptors	32	32k (In simulation we use only 320 buffers)	1 MB		
Queue Descriptors	16	256 (16 ports x 16 classes per port)	4k		
CSIX RX Reassembly contexts	32	1024	32k		
Buffers	2048	32k		64 MB	
CSIX RX to Statistics ring	12	512/3 (the size of the ring is 512 long words, but each entry enqueued uses 3 long words. Therefore the total number of entries is 512/3 = 170)			2k
QM Q-Array entries	N/A	16			
DRR Scheduler Queue Structures	32	16 ports * 256 queues	131 KB		
DRR Scheduler Round Counters	4	16 ports * 4K rounds	262 KB		
Buffer Free list Q-Array entry	N/A	4			

5.6 Interfaces Between the Various Microblocks

This section describes the interfaces between the different microblocks in the ingress and egress processors for this application.

5.6.1 Packet RX—First ME to Second ME

The interface between the first Packet RX microengine and second Packet RX microengine is a next neighbor (NN) ring. Table 5-6 describes each entry in the NN ring—which is five long words.

Table 5-6. Five-Word NN Ring Entry (Packet RX—First ME to Second ME)

LW	Bits	Size	Field	Description
0	31:0	32	dram_handle	DRAM address where the m-packet should be stored
1	31:0	32	curr_buf_handle	Buffer handle of the current buffer of the packet (only valid if eop_flag is 1)
2	31:0	32	sop_buf_handle	Buffer handle of the SOP buffer of the packet (only valid if eop_flag is 1)
3	31:16	16	sop_buf_size	SOP buffer size in bytes (only valid if eop_flag is 1)
	15:15	1	eop_flag	Bit indicating if this is the last m-packet of the packet
	14:8	7	rbuf_elem	RBUF element number containing the m-packet
	7:0	8	byte_count	Number of bytes to copy from RBUF to DRAM
4	31:16	16	input_port	Input port on ingress processor (only valid if eop_flag is 1)
	15:0	16	packet_size	Total packet size across buffers in bytes (only valid if eop_flag is 1)

5.6.2 Packet RX and Packet Processing Microengines

The interface between the Packet RX microblock and the packet processing microengines is a scratch ring. Table 5-7 describes each entry in the scratch ring—which is three long words.

The format depends on whether the packet fits in one buffer or not. In the case of packets that span across multiple buffers, some of the packet descriptor information is written to SRAM and the rest to the scratch ring. In the case of packets that fit into a single buffer, all the information is packed into the scratch ring eliminating one read/write to SRAM in the critical path. Bit 31 of LW0 (EOP bit of the handle) is used to detect if a packet spans across multiple buffers. If this bit is set (implying that the buffer is a SOP/EOP buffer), then the packet is contained in a single buffer.

This interface is used for packets that fit entirely in one buffer.

Table 5-7. Three-Word Scratch Ring Entry —Packets fit on one Buffer

LW	Bits	Size	Field	Description
0	31:0	32	dl_buffer_handle	Buffer Handle for the SOP Descriptor
1	31:16	16	input_port	Input port on ingress processor
	15:12	4	free_list_id	Free list ID for buffer
	11:8	4	rx_stat	Receive Status Flag
	7:0	8	header_type	Type of header at offset bytes into the packet
2	31:16	16	buffer_size	Buffer size in bytes
	15:0	16	offset	Offset of the start of data in the SOP buffer in bytes

This interface is used for packets that require more than one buffer.

Table 5-8. Three-Word Scratch Ring Entry —Packets Require more than one Buffer

LW	Bits	Size	Field	Description
0	31:0	32	dl_buffer_handle	Buffer Handle for the SOP Descriptor
1	32:0	32	dl_eop_buffer_handle	Buffer Handle for the EOP Descriptor
2	31:16	16	packet_size	Total packet size across buffers in bytes
	15:0	16	offset	Offset of the start of data in the SOP buffer in bytes

5.6.3 Packet Processing Microengines and Statistics

Packet Processing Microengines and Statistics interface is a scratch ring. Table 5-9 describes each entry in the scratch ring—which is three long words.

Table 5-9. Three-Word Scratch Ring Entry—Packet Processing Microengines and Statistics

LW	Bits	Size	Field	Description
0	30:16	16	MOP_EOP_buf_size	Size in bytes of all MOP buffers and the EOP buffer of the packet
0	0:15	16	Queue Number	Queue Number
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

5.6.4 Statistics and CSIX Scheduler

The Statistics and CSIX Scheduler interface is a next neighbor (NN) ring. Table 5-10 describes each entry in the NN ring—which is three long words.

Table 5-10. Three-Word NN Ring Entry (Statistics and CSIX Scheduler)

LW	Bits	Size	Field	Description
0	30:16	16	Packet cell count	Sum of all buffer cell counts belonging to the packet
0	0:15	16	Queue Number	Queue Number
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

5.6.5 CSIX Scheduler and Cell Queue Manager

The CSIX Scheduler and Cell Queue Manager interface is a next neighbor ring. Table 5-11 describes each entry in the NN ring—which is three long words.

Table 5-11. Three-Word NN Ring Entry (CSIX Scheduler and Cell Queue Manager)

LW	Bits	Size	Field	Description
0	30:16	16	Dequeue Queue #	Queue number from which to dequeue. Zero implies no dequeue
0	0:15	16	Enqueue Queue #	Queue number on which to enqueue. Zero implies no enqueue
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

5.6.6 Cell Queue Manager and CSIX TX

The Cell Queue Manager and CSIX TX interface is a next neighbor ring. CSIX Transmit is a two-microengine context pipe-stage. The cell queue manager writes to the NN ring of the first CSIX TX microengine. Table 5-12 describes each entry in the NN ring—which is two words.

Table 5-12. Two-Word NN Ring Entry (Cell Queue Manager and CSIX TX)

LW	Bits	Size	Field	Description
0	31:16	16	Reserved	Reserved
0	15:0	16	Queue Number	Queue Number
1	31:0	32	Buffer Handle	Buffer Handle currently being transmitted for queue

5.6.7 CSIX TX—First ME to Second ME

The interface between the first CSIX TX microengine and second CSIX TX microengine is a next neighbor ring. Table 5-13 describes each entry in the NN ring—which is eight long words.

Table 5-13. Eight-Word NN Ring Entry (CSIX TX—First ME to Second ME)

LW	Bits	Size	Field	Description
0	31:0	32	Tx_request0	Same as LW0 from Cell Queue Manager to CSIX TX
1	31:0	32	Tx_request1	Same as LW1 from Cell Queue Manager to CSIX TX
2	31:0	32	dram_handle	DRAM address where CSIX cell is stored
3	31:24	8	cell_count_remaining	Number of cells remaining in the current buffer
	23:18	6	Reserved	Reserved
	17:17	1	MOP_EOP_flag	If MOP_EOP, set to 1, else 0
	16:16	1	SOP_EOP_flag	If SOP and EOP, set to 0, else 1
	15:0	16	payload_length	Length of CSIX cell payload in bytes

Table 5-13. Eight-Word NN Ring Entry (CSIX TX—First ME to Second ME) (Continued)

LW	Bits	Size	Field	Description
4	31:0	32	prepend_header0	LW0 of CSIX cell pre-pend header
5	31:0	32	prepend_header1	LW1 of CSIX cell pre-pend header
6	31:0	32	prepend_header2	LW2 of CSIX cell pre-pend header
7	31:0	32	prepend_header3	LW3 of CSIX cell pre-pend header

5.6.8 CSIX TX (Second ME) and Freelist Manager

The interface between the second CSIX TX microengine and the Freelist Manager is a next neighbor ring. Table 5-14 describes each entry in the NN ring—which is eight long words.

Table 5-14. One-Word NN Ring Entry

LW	Bits	Size	Field	Description
1	31:0	32	Buffer Handle	Buffer Handle to be freed by the Freelist Manager

5.6.9 Freelist Manager and Packet Rx (First ME)

The interface between the Freelist Manager and the first Packet Rx microengine is a next neighbor ring. Table 5-15 describes each entry in the NN ring—which is eight long words.

Table 5-15. One-word NN Ring Entry

LW	Bits	Size	Field	Description
1	31:0	32	Buffer Handle	Buffer Handle that is allocated by the Freelist Manager

5.6.10 CSIX RX and Statistics

The CSIX RX and Statistics interface is a scratch ring. Table 5-16 describes each entry in the scratch ring—which is three words

Table 5-16. Three-Word Scratch Ring Entry (CSIX RX and Statistics)

LW	Bits	Size	Field	Description
0	30:16	16	Packet Size	Packet Size
0	15:12	4	Port Number	Output Port Number
0	11:0	12	Queue Number	Queue Number
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

5.6.11 Statistics and Packet Scheduler

Table 5-17 shows the Statistics and Packet Scheduler interface, which is a Next Neighbor ring.

Table 5-17. Three-Word NN Ring Entry (Statistics and Packet Scheduler)

LW	Bits	Size	Field	Description
0	30:16	16	Reserved	Reserved
0	15:0	16	Packet Size	Packet Size
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:16	16	Port Number	Output Port Number
2	31:0	16	Queue Number	Queue Number

5.6.12 Packet Scheduler and Queue Manager

The interface between the Queue Manager and the Packet Scheduler is a Next Neighbor Ring. Table 5-18 describes each entry in the NN ring—which is three long words.

Table 5-18. Three-word NN Ring Entry (Queue Manager and Packet Scheduler)

LW	Bits	Size	Field	Description
0	30:16	16	Dequeue Queue #	Queue number from which to dequeue. Zero implies no dequeue
0	0:15	16	Enqueue Queue #	Queue number on which to enqueue. Zero implies no enqueue
1	31:0	32	SOP Buffer Handle	Buffer Handle for SOP Descriptor
1	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

5.6.13 Queue Manager and TX Helper

The interface between the Queue Manager and the TX helper is a Next Neighbor ring. Table 5-19 describes each entry in the NN ring—which is one word:

Table 5-19. Two-Word NN Ring Entry (Queue Manager and Packet TX)

LW	Bits	Size	Description
0	31:4	28	Reserved
0	3:0	4	Port number
1	31:24	8	Reserved
1	23:0	24	Pointer to SOP buffer descriptor in SRAM in long words (Same as bits 0:23 of buffer handle)

5.6.14 TX Helper and Packet TX

The interface between the TX helper and the Packet Transmit is a Next Neighbor ring. Table 5-20 describes each entry in the NN ring—which is one word:0

Table 5-20. One-Word NN Ring Entry (Queue Manager and Packet TX)

LW	Bits	Size	Description
0	31:31	1	Valid bit
0	30:28	3	Reserved
0	27:24	4	Port number
0	23:0	24	Pointer to SOP buffer descriptor in SRAM in long words (Same as bits 0:23 of buffer handle)

5.6.15 Packet TX—First ME to Second ME

The interface between the first microengine and second microengine of Packet Transmit is a Next Neighbor ring. Table 5-21 describes each entry in the NN ring—which three words.

Table 5-21. Three-Word NN Ring Entry (Packet TX—First ME to Second ME)

LW	Bits	Size	Description
0	31:0	32	Pointer to meta data (used to free buffer
1	31	1	Bit is clear if the m-packet is sop
	30	1	Bit is clear if the m-packet is eop
	29:0	29	Offset of payload to be transmitted
2	31:0	32	Payload size to be transmitted

If the m-packet is non-stop, then 3 more long words are included on the ring.

Table 5-22. Three-Word NN Ring Entry (for Non-stop m-packet)

LW	Bits	Size	Description
3	31:0	32	Bytes from previous buffer to be prepended to the current buffer
4	31:0	32	Exe_stat_flag: information about various condition flags
5	31:0	32	Partially created transmit control word

5.7 Porting from IXP2400 to IXP2800

This section describes how the POS IPv4 Forwarding application for the Intel® IXP2400 Network Processor was ported to the Intel® IXP2800 Network Processor.

5.7.1 IXP2400 and IXP2800 Processing Requirement Comparison

Table 5-23 shows a comparison of IXP2400 and the IPX2800 processing requirements.

Table 5-23. Comparison of IXP2400 and the IPX2800 Processing Requirements

Item	IXP2400	IXP2800
Clock Frequency	600 MHz	1400 MHz
SRAM Frequency	200 MHz	200 MHz
SRAM Channels	2	4
SRAM Read bandwidth	1600 MB/s = 200 * 2 * 4	3200 MB/s = 200 * 4 * 4
SRAM Write bandwidth	1600 MB/s	3200 MB/s
DRAM frequency	150 MHz DDR	1066 MHz RDRAM (IXP2800 drives at 1018.18)
DRAM channels	1 (4 banks)	3 (4 banks per channel)
DRAM Read/Write bandwidth	2500 MB/s	6109 MB/s = 1018 * 3 * 2
DRAM efficiency	60%	72%
Effective DRAM bandwidth	1500 MB/s	4423 MB/s = 72% * 6109
Number of microengines	8	16
Data rate	OC-48 (6.12 mpps)	OC-192 (24.5 mpps)
Instruction budget per microengine per 49 byte POS min packet	97 = 600/6.12	57 = 1400/24.5

As Table 5-23 shows, the IXP2800 at OC-192 data rates needs to process 4 times as many packets as the IXP2400. The IXP2800 has 4.6 (= 1400/600 * 16/8) times the processing capability of the IXP2400. However for a single microengine in the pipeline (e.g. the Queue Manager), the instruction budget for a 49 byte POS min-packet for the IXP2800 is only 57 cycles compared to 97 cycles for the IXP2400. At the same time, the IXP2800 has only twice the SRAM bandwidth and approximately three times the DRAM bandwidth of the IXP2400.

This has the following implications:

- For the same application, the IXP2400 at OC-48 data rates has almost twice as much SRAM memory bandwidth available as the IXP2800 at OC-192 data rates. Therefore SRAM usage must be optimized as much as possible for the IXP2800 application.
- For the same application, microblocks executing on a single microengine (e.g. the driver blocks such QM, scheduler, etc) must be optimized to run in 57 cycles or the design of these blocks must be modified so they can execute on multiple microengines. The packet processing microengines on the other hand have more compute cycles running on the IXP2800 than the IXP2400.

5.7.2 Optimizations for the IXP2800

This section describes various optimizations made to the IXP2400 application to port it to OC-192 data rates on the IXP2800.

5.7.2.1 Optimizing SRAM Memory Bandwidth Usage

A major part of the optimization effort involved moving data structures across the SRAM channels to achieve better and more uniform distribution of SRAM bandwidth usage per channel.

Table 5-24 compares the location of data structures between the IXP2400 and IXP2800 applications.

Table 5-24. Data Structure Location Comparison of the IXP2400 and IXP2800 Applications

Data	IXP2400 SRAM channel	IXP2800 SRAM channel
Buffer metadata LW0	0	0
Buffer metadata LW1-7	0	3
Queue descriptors	0	0
Packet RX counters	1	1
Next hop table	0	2
Trie table	1	1
Directed Broadcast table	1	2
Control block information	1	In scratch memory
IPv4 counters	1	In scratch memory
CSIX TX contexts	0	0

Also the IPv4 Next Hop data structure was compressed from four long words to two long words. The compressed structure is described in the *Intel® Internet Exchange Architecture Software Building Blocks Developer's Manual*, Section 24.6.3, "Next Hop Information" on page 405.

5.7.2.2 Splitting the Packet Descriptor Across Channels

At OC-192 data rates, the channel used for queuing is completely utilized by the Q-Array. Therefore the per-packet descriptor—that is, metadata was split across channel 0 and 3. The first word (LW0) is used as a next pointer by the Q-Array hardware for maintaining the link list. This word is kept in channel 0. The remaining fields are moved to channel 3. This change is hidden from the microblocks via the `dl_meta_xxx()` macros.

5.7.2.3 Splitting the RX/TX Driver Blocks to Run on Multiple Microengines

To meet the 57 cycle budget for OC-192 POS, the RX and TX blocks for POS and CSIX were modified to run on multiple microengines. In the case of Packet RX, CSIX TX and Packet TX, the two microengines run as a context pipeline connected by a Next Neighbor ring. In the case of CSIX RX, the two microengines run in parallel executing the same code.

Currently these blocks all support a TWO_ME compile time option that may be used to run them on two microengines and achieve which may be used to run them on two microengines and achieve OC-192 line rates for min POS packets.

5.7.2.4 Moving Data Structures to Local Memory

As yet another memory usage optimization, the Directed Broadcast table used in IPv4 may be moved to local memory and updated only periodically.

Note: Future release will be implemented to support this.

5.7.2.5 Optimizing the Packet Buffer Freelist

One optimization to improve the Q-Array performance is to use a Next Neighbor ring between the TX and RX microengines for allocating and freeing buffers. The general idea is to populate this ring with 128 buffer handles initially. When the Receive microengine needs a buffer, it first attempts to allocate it from the Next Neighbor ring. If the ring is empty, it allocates it from the Q-Array buffer free list. Similarly when the transmit code needs to free a buffer handle, it first attempts to write it to the Next Neighbor ring. If the ring is full, then it frees to the Q-Array free list.

5.7.2.6 Using NN Ring Instead of Scratch Ring for Communication

Throughout the design, an effort has been made to use Next Neighbor rings where possible to minimize use of scratch bandwidth.

5.7.2.7 New Design for the Scheduler and Queue Manager

The design for the scheduler and queue manager blocks had to be modified for the IXP2800. This change was driven by:

- The need to fit these blocks into 57 cycles
- The need to avoid invalid dequeues (dequeue requests to queues with no data). This is critical to meet line rate.

The new design essentially places the scheduler after the packet processing microengines in-line and before the Queue Manager. It receives the enqueue requests from the packet processing microengines and generates the dequeue requests. This allows it to keep track of the number of entries in a queue thereby avoiding invalid dequeues. To keep track of which queues have data, the scheduler uses a link list of active queues in local memory. This is more efficient (requires less instructions) than the bit-vectors used for the IXP2400 design. It however can support fewer queues than the bit-vector based design.

The Queue Manager is now very much simplified since:

- It no longer keeps track of queue count and queue transitions and does not need to send any transition messages to the scheduler.
- It reads all enqueue and dequeue requests from a Next Neighbor ring. Therefore the code no longer requires multiple phases.

OC-192 POS IPv4 MPLS Application 6

This section describes the design of an IPv4 MPLS Forwarding application using the Intel® IXP2800 Network Processor. Two half-duplex IXP2800 processors are used to implement a POS line card at OC-192 data rates that interfaces to a CSIX switch fabric. This section provides a high-level design overview and lists the different software components used to build this application. It focuses only on the fast path or microengine components of the design. The Intel XScale® core components for this application are described in *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*.

The application described in this chapter is supported on the Intel® IXDP2800 Advanced Development Platform.

This application is modified from the OC-192 POS IPv4/IPv6 application with the IPv6 block being removed and the MPLS microblocks being added. Since the changes occur in the Ingress side, this section describes the microblocks in the Ingress side only. Details of the microblocks in the Egress side (which are exactly the same for both applications) can be found in the OC-192 POS IPv4/IPv6 application.

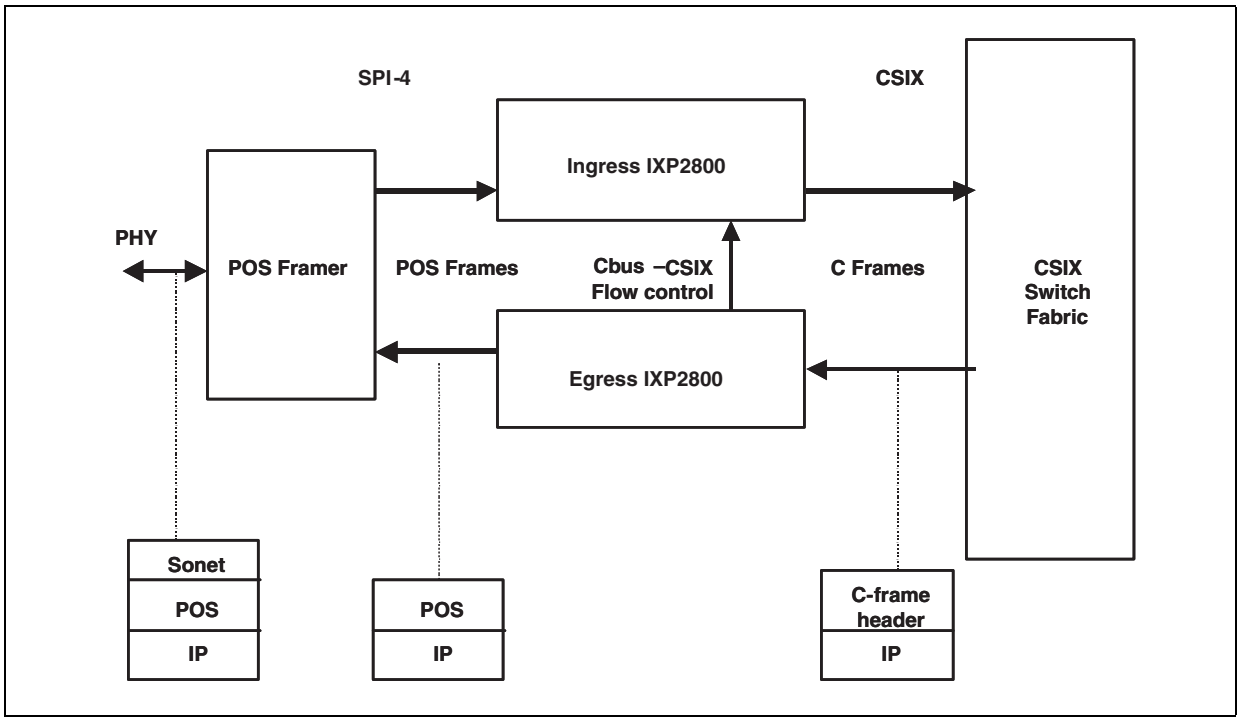
6.1 Hardware Overview

Figure 6-1 illustrates an example hardware configuration for OC-192 POS line card with CSIX fabric. The figure shows two IXP2800 processors in a typical CSIX full duplex configuration. In this configuration, the two IXP2800 processors are identified as the ingress processor (receives from the Media interface and transmits to the CSIX Fabric) and the egress processor (receives from the CSIX Fabric and transmits to the Media interface).

The Ingress IXP2800 receives POS frames that carry IPv4 or MPLS datagrams. The frames are assembled into IPv4 or MPLS packets and the Layer-2 (PPP) headers are removed after being classified. If it is IPv4 packet, a Longest Prefix Match (LPM) lookup is performed based on IPv4 header. If it is MPLS packet, an Incoming Label Map (ILM) lookup is performed based on MPLS labels. Packets are then segmented into CSIX C-Frames and transmitted to the CSIX fabric. The result of the LPM/ILM lookup determines which IXP2800 connected to the fabric receives the packet, and which port on that IXP2800 the packet is transmitted on.

The Egress IXP2800 receives CSIX C-Frames from the fabric and reassembles these into IPv4 or MPLS datagrams. The Layer-2 (PPP) headers are added and the packets are transmitted over the appropriate port.

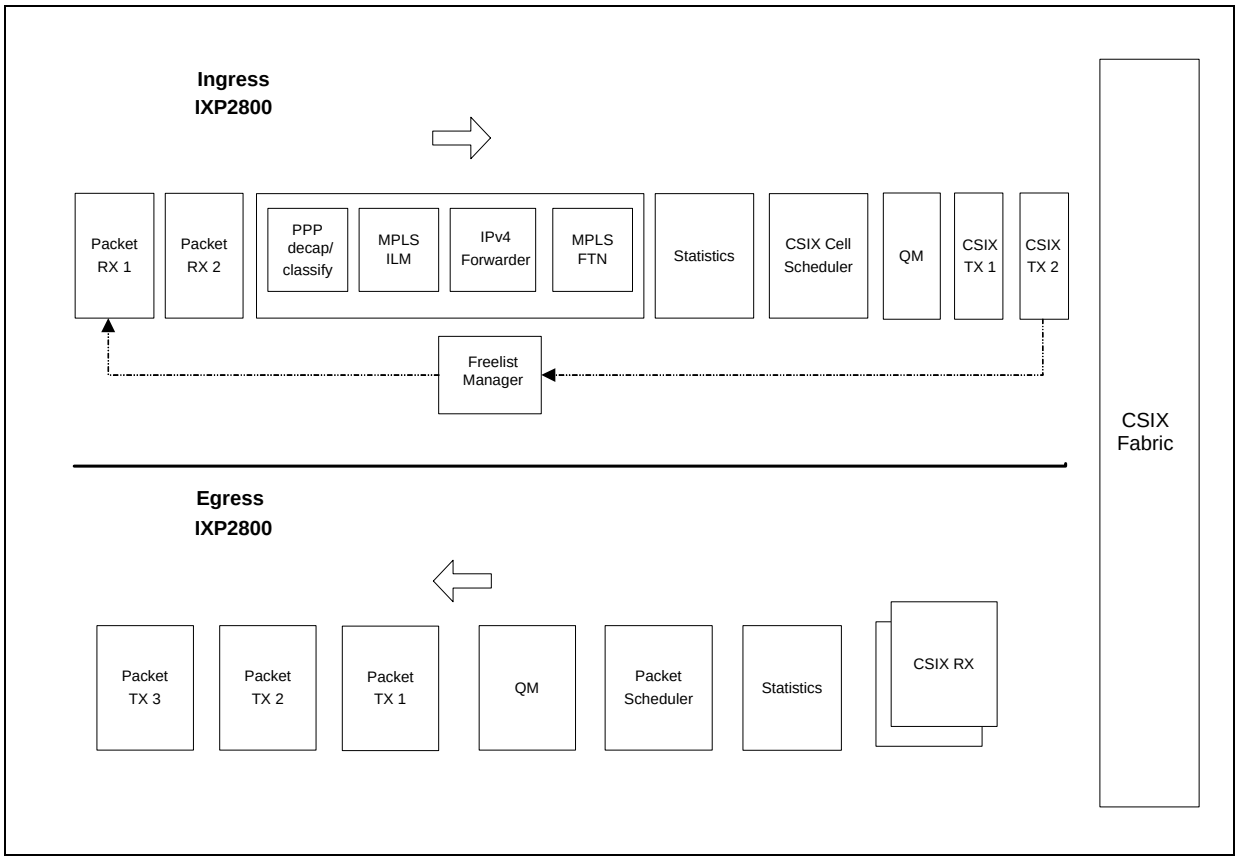
Figure 6-1. Example Hardware Configuration for OC-192 POS Line Card with CSIX Fabric



6.2 Software Overview

Figure 6-2 illustrates the microblocks needed to implement an OC-192 POS IPv4 MPLS Forwarding application. The design for this application is based on the guidelines specified by the IXA Portability Framework in the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*. The driver microblocks (Receive, Transmit, Scheduler, QM, Statistics and FreeListManager) run on different microengines from the packet processing code.

Figure 6-2. Microblocks for an OC-192 POS IPv4 MPLS Forwarding Application



6.2.1 Data Flow for the Ingress

6.2.1.1 Packet RX

The Packet RX microblock runs on two microengines in a context pipeline connected by a Next Neighbor ring. The Packet RX microblock for the IXP2400 (Section 2.2.1.1, “Packet RX” on page 25) has been extended such that as a compile time option it now runs on two microengines.

This microblock performs frame-reassembly on the mpackets coming in on the POS media interface. It reassembles and writes the packet data to a buffer in DRAM and queues the packet buffer handle on a ME-ME scratch ring for processing by the packet processing microengines. It also sets up per- packet meta information (offset, size etc) which are passed on either in a descriptor in SRAM or in the ME-ME scratch ring itself. Up to 16 virtual ports are supported and the re-assembly context for all these ports is kept in local memory. To maintain packet sequencing, the threads execute in strict order. The microblock is written such that it supports up to 16 virtual ports, but one or more of these may be unused. This allows the same microblock to support different configurations such as Quad OC-48, 16 OC-12 or a single OC-192 port.

In this application, the packets reassembled are PPP frames containing IP datagrams. RFC 2615 defines the Packet Over SONET specification and refers to RFC 1661 (PPP) and RFC 1662 (PPP in HDLC-like framing). PPP framing including header validation, FCS generation and computation and byte stuffing are handled by the POS framer (IXF 18101).

Since POS packets may be up to 9k bytes, some large packets may be stored in multiple buffers chained together as a link-list. The buffer handles for the first and last packet in the chain are queued in the scratch ring.

From the Packet RX block, the packet is passed on to an application specific system microblock (`DL_Sink[]`). This microblock checks if the packet has been marked to be dropped (`IX_DROP`) or sent to the Intel XScale® core (`IX_EXCEPTION`). If not, it queues the packet buffer handle and associated packet meta data into the scratch ring for the next stage in the pipeline.

6.2.1.2 Packet Processing Microengines (PPP Decap/Classify + MPLS ILM + IPv4 Forwarder + MPLS FTN)

The PPP decapsulation/classify microblock runs along with the IPv4 and MPLS microblocks on 8 microengines or 64 threads. These microblocks (except MPLS blocks) are identical to the ones used for the IXP2400 POS application described in Section 2.2.1.2, “PPP Decapsulation and Classify” on page 25 and Section 2.2.1.3, “IPv4 Forwarder” on page 26.

An application specific system source microblock on each thread dequeues packet buffer handles from the scratch ring. This source block (`DL_Source[]`) is a system microblock implicit in the dispatch loop. It reads in the packet meta information (the packet descriptor) and populates the dispatch loop state. It also reads in 24 bytes of the packet header from DRAM into transfer registers and then caches them in local memory. Since it is important to maintain packet sequencing, the threads in the microblock execute in strict order to dequeue from the scratch ring. This implies that the first thread on microengine 1 dequeues the first packet, signals the next thread to perform the dequeue and so on. From this block, the packet goes to the PPP decapsulation/classify microblock.

The PPP decapsulation/classify microblock removes the layer-2 PPP header from the packet by updating the offset and size fields in the packet descriptor. Based on the PPP header, it also classifies the packet into IPv4, IPv6, MPLS, or PPP control packet (LCP, IPCP). If the packet is a PPP control packet, it is marked as an exception packet to be sent to the Intel XScale® core (`IX_EXCEPTION`). Otherwise the packet is sent down the microengine pipeline for further processing. In this application, the dispatch loop silently drops packets classified as IPv6.

If the packet is an MPLS packet, the MPLS ILM (Incoming Label Map) forwarder microblock forwards the packet based on the MPLS labels (per RFC 3031, 3032). First the top MPLS label is checked against reserved values. Then it is mapped to an entry in the ILM NHLFE table (Next Hop Label Forwarding Entry) where information as to how the label is processed and how the packet is forwarded is obtained. This microblock handles LSR and Egress LER cases (SWAP, POP, POP_FORWARD and SWAP_PUSH operations). The result of ILM is a Next Hop ID, a fabric blade id, and an output port which are all stored in the packet metadata for later use by the egress side. The MPLS ILM forwarder microblock supports two label space modes defined by the following compilation switches:

- PER_PLATFORM_LABEL_SPACE (set by default): label ranges (min, max) and table base offset for the whole system are configured at initialization time.
- PER_INTERFACE_LABEL_SPACE: label ranges (min, max) and table base offset are configured at initialization time on a per-interface basis.

The MPLS ILM forwarder microblock reads label space values from SRAM and stores them in local memory at initialization time.

The IPv4 forwarder microblock validates the IP header per RFC 1812. If the validity checks fail, then the packet is set up to be dropped as specified in *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*. Otherwise a Longest Prefix Match (LPM) is performed on the IPv4 header. The result is an IPv4 Next Hop ID, a fabric blade id (identifying a unique IXP2800 on the fabric) and an output port identifying the output port on the egress IXP2800. The Next Hop ID is passed over the CSIX fabric to an Egress IXP2800 where it is used to look up information about the Layer-2 header to be prepended to the packet buffer. The output port is also passed over the CSIX fabric to the egress IXP2800 and is used to transmit over the appropriate port. All three fields are stored in the packet meta data—that is, the packet descriptor. If no match is found, then the packet is set up to be sent up to the Intel XScale® core for further processing as specified in *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*. Packets are also sent to the core in a number of other cases, for example when the packet is destined for a local interface or is to be fragmented.

From the IPv4 forwarder block, the packet is passed on to an application specific system microblock (`DL_QM_Sink[]`) if the Next Hop ID type indicates IPv4 type or to the MPLS FTN microblock if the Next Hop ID type indicates MPLS type.

The MPLS FTN (FEC-To-NHLFE) microblock maps the FEC (Forwarding Equivalence Classes) to an entry in the FTN NHLFE (Next Hop Label Forwarding Entry) table. The Next Hop ID generated from the IPv4 forward block is used as an FEC. This block handles Ingress LER cases (PUSH operation). The packet is encapsulated with up to 4 MPLS labels and the packet header is changed to MPLS type. Values of next hop id, fabric blade id and output port are obtained from the entry and stored in the packet metadata. The packet is passed on to an application specific system microblock (`DL_QM_Sink[]`).

This application specific system microblock checks if the packet is to be dropped or sent to the Intel XScale® core. If not, it sends an enqueue request to the Statistics microengine over a scratch ring. The `DL_QM_Sink[]` microblock also writes the cached packet header to DRAM and the packet meta information to SRAM.

6.2.1.3 Statistics Microblock

This microblock runs on a single microengine. It is currently a place holder for statistics handling. It is anticipated that this microblock is used to manage per-flow statistics for future MPLS and DiffServ applications.

Note: The design for handling statistics will be described in future revisions of the document.

The statistics microengine interfaces to the IXP2800 CSIX Fabric Scheduler microblock via a Next Neighbor ring, passing it the packet enqueue requests received from the packet processing microengines. It also computes the total cell count of every packet enqueued and passes it to the scheduler. In addition, it also handles dropping of large packets that are stored in multiple buffers.

6.2.1.4 CSIX Scheduler

The CSIX scheduler runs on a single microengine and schedules c-frames into the CSIX fabric. This microblock is significantly different from the one currently used on the IXP2400. It has been optimized to run in 57 cycles which is the min POS packet instruction budget. Also it is placed in

the packet processing pipeline before the queue manager, allowing it to keep track of enqueue and dequeue transitions correctly and without any latency. Unlike the IXP2400 version which handles 1024 VoQs (Virtual Output Queues), the design used for the IXP2800 supports 256 VoQs.

The scheduling algorithm implemented is Round Robin among the ports on the fabric and Weighted Round Robin among the queues on a port. Since this is not a QoS application and there is only one queue per port, the Weighted Round Robin scheduling degenerates to round robin scheduling. Other applications, for example, IP DiffServ may use the WRR functionality. The scheduling and transmit is done a cframe at a time.

The CSIX scheduler handles the following:

- Flow control messages from the fabric. These messages are sent by the fabric to the egress IXP2800, which sends them on the c-bus to the ingress IXP2800. If the fabric asserts Xoff on a particular VoQ (Virtual Output Queue), the scheduler stops scheduling for the queue.
- Packet enqueue requests from the previous microengine. It uses this information to update a list of active queues (queues with data) and to track queue transitions (empty to non-empty and vice-versa). A queue is scheduled only if there is data in the queue. The enqueue requests are passed on via Next Neighbor ring to the Queue Manager.
- MSF Transmit State Machine. The scheduler monitors how many packet cframes are in the pipeline and if it exceeds a certain threshold, it stops scheduling.

During each loop, the scheduler also:

- Checks its list of active queues (queues with data). Picking up from where it left off in the last iteration, it finds the next queue to schedule.
- It then sends a dequeue message to the Queue Manager to dequeue the head of that queue. The Queue Manager dequeues a cell (cframe) from the head of the queue and sends a transmit request to the CSIX TX microblock.

6.2.1.5 Cell Based Queue Manager (Cell QM)

The Queue Manager (QM) is a driver microblock that runs on a single microengine. This microblock is significantly different from the one currently used in the IXP2400 application. It has been optimized to run within 57 cycles which is the instruction budget for a min POS packet at OC-192 data rates. The key difference is that in the IXP2800 design, the scheduler keeps track of the queue size and queue transitions. This considerably simplifies the Queue Manager which no longer has to support this functionality.

The QM manages enqueue and dequeue operations on the transmit queues which are implemented using the hardware SRAM link lists. It accepts enqueue requests from the scheduler via a Next Neighbor ring. The enqueue requests are on a per-packet basis. The dequeue requests come are on a per-cell basis where a cell is a CSIX cframe.

The threads on the QM microengine execute in strict order using local inter-thread signaling. SRAM Queue Array entries are cached in the SRAM controller and the CAM is used for managing the tags for these. To maintain coherence among threads, folding is used.

6.2.1.6 CSIX TX

The CSIX Transmit microblock runs on two microengines in a context pipeline connected by a Next Neighbor ring. The CSIX Transmit microblock for the IXP2400 (Section 2.2.1.6, “CSIX TX” on page 27) has been extended so that as a compile time option it now runs on two microengines.

This microblock receives transmit messages from the queue manager via a Next Neighbor ring. With each transmit request, the microblock moves a cframe into a TBUF, which is then transmitted into the fabric by the MSF Transmit State Machine.

Every request has an associated packet, which is being segmented into cframes. The associated segmentation state for the packet and the packet metadata is cached in local memory and is looked up using the CAM. The TX microblock adds the CSIX header onto the cframe along with the packet data. Along with the CSIX header, a Traffic Manager (TM) header is also added per cframe carrying extra information (destination Layer-2 port id, input blade id, sequence number, next-hop id etc.) about the packet to be passed to the Egress IXP2800. In addition, the flow id, class id, input port and some other fields from the metadata are passed along to the Egress IXP2800 using a per-packet header pre-pended to the start of the first c-frame of each packet.

6.2.1.7 Free List Manager

The Free List Manager service microblock runs on a single microengine. Refer to Section 5.2.1.7, "Freelist Manager" on page 70 inChapter 5, "OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application" for details of this microblock.

6.2.2 Data Flow for the Egress

For details, refer to Section 5.2.2, "Data Flow for the Egress IXP2800" on page 71 in Chapter 5, "OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application."

6.3 Performance Characterization

The Intel® IXP2800 Network Processor operates at 1400 MHz. For a min POS packet of 49B, the packet inter-arrival time at OC-192 line rate is 57 ME cycles. In order to maintain line rate for min packets, each stage of the pipeline cannot exceed this budget. In other words, each stage of the pipeline needs to retire a packet every 57 cycles.

Table 6-1 summarizes the performance analysis for the POS pipeline.

Table 6-1. Performance Analysis for the POS Pipeline

OC-192c line rate assuming 3% SONET overhead	9.62 Gigabits/sec
Min POS packet size	49 bytes (40 byte TCP/IP, 2 bytes Address and Control, 2 byte PPP header, 4 byte FCS and 1 byte flag)
Packet Throughput for min packets	24.56 million packets/sec = (9.62 / (49*8)) * (10**9)
IXP2800 clock frequency	1400 MHZ
Inter-packet arrival time for min packets	1400/6.14 = 57 cycles
Compute cycles per packet for a single microengine	57
Latency per packet for a single microengine	57 * 8
Compute cycles per packet for n microengines running in parallel	57*n
Latency per packet for n microengines running in parallel	57*8*n

6.4 Ingress System Resource Allocation

Table 6-2 shows the system resources mapped for the Ingress IXP2800. This mapping reflects the system defaults and may be changed. The allocation of microengines is done such that it optimizes the performance of this specific application and may be changed for other applications.

Table 6-2. System Resources Mapped for the Ingress IXP2800

Microblock	ME #	Communication Mechanism with previous stage
Packet RX	ME 1:3, 1:4	Auto-push status from MSF
Layer2 decapsulation/Classify + MPLS ILM + IPv4 Forwarder + MPLS FTN	ME 0:0, 0:1, 0:2, 0:3, 0:4, 1:5, 1:6, 1:7	Scratch ring
Statistics	ME 0:5	Scratch ring
CSIX Scheduler	ME 0:6	NN ring
Queue Manager	ME 0:7	NN ring
CSIX TX	ME 1:0, 1:1	NN ring
FreeListManager	ME 1:2	NN ring
Headroom	0 microengines	

The physical assignment of function to microengine is important since it not only affects when the next neighbor registers and signaling can be utilized, but it also affects the utilization of the internal Command bus and S-Push/Pull buses. This assignment attempts to balance the usage of the Command bus and S-Push/Pull buses across the two clusters.

The IXP2800 supports four SRAM channels and three DRAM channels. Table 6-3 shows the SRAM, DRAM and scratch utilized for this application. These values are defined in a system header file `dl_system.h` and may be changed as needed.

Table 6-3. SRAM, DRAM, and Scratch Utilization for Ingress IXP2800

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Buffer Descriptors	32	32k (In simulation, we use only 320 buffers)	1 MB		
Buffers	2048	32k		64 MB	
Queue Descriptors	16	256 (1 per VOQ)	4K		
CSIX TX contexts	32	256 (1 per VOQ)	8k		
Trie Table	64 (The root Trie table requires at least 257k to support hi64k and hi256 tables. In addition each node requires 64 bytes. These nodes are added as needed)	See note in previous column. Assuming 256k routes, approximately 128k nodes are needed	8MB		
Route Table (Next Hop Information)	8	Assuming 4k next hops	32k		
IPv4 statistics	4	16			64

Table 6-3. SRAM, DRAM, and Scratch Utilization for Ingress IXP2800 (Continued)

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
MPLS FTN NHLFE	32	64k	2M		
MPLS ILM NHLFE	32	64k	2M		
MPLS per-context header caching	64 (local memory)	8			
Packet RX statistics	4	16*16	1024		
IPv4 Directed Broadcast Table	32 (local memory)	64			
Ring from Packet RX to packet processing pipeline (IPv4+Layer2 Decap/Classify)	12	4k/3			4k
IPv4 to Statistics ring	12	2k/12			2k
QM Q-Array entries	N/A	16			
Buffer Free list Q-Array entry	N/A	4			

6.5 Egress System Resource Allocation

Please refer to Section 5.5, “Egress System Resource Allocation” on page 74 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application.”

6.6 Interfaces Between the Various Microblocks

Please refer to Section 5.6, “Interfaces Between the Various Microblocks” on page 75 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application.”

6.7 Application Optimizations

This section points out optimizations made to the Ingress side of the MPLS application to achieve OC-192 data rates on IXP2800 besides those already mentioned in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application.”

6.7.0.1 Optimizing SRAM Memory Bandwidth Usage

A major part of the optimization effort involved moving data structures across the SRAM channels to achieve better and more uniform distribution of SRAM bandwidth usage per channel.

Table 6-4. Data Structure Allocations

Data	IXP2800 SRAM channel
Buffer metadata LW0	0
Buffer metadata LW1-7	1
Queue descriptors	0
Packet RX counters	1
Next hop table	3
Trie table	3
MPLS ILM NHLFE	2
MPLS FTN NHLFE	2
IPv4 counters	In scratch memory
CSIX TX contexts	0

Also the IPv4 Next Hop data structure was compressed from four long words to two long words. The compressed structure is described in the *Intel® Internet Exchange Architecture Software Building Blocks Developer's Manual* Section 24.6.3, "Next Hop Information" on page 405.

6.7.0.2 Moving Data Structures to Local Memory

As yet another memory usage optimization, the Directed Broadcast table used in IPv4 is moved to local memory and updated only periodically.

6.7.0.3 Caching Packet Header in Local Memory

Each thread of a microengine is allocated up to 16 local memory longwords (LW) to use for packet header caching. As packets can grow and shrink in sizes when entering and exiting MPLS domain, the packet headers are cached in the 5th LW of the memory cache. This allows up to 4 MPLS labels to be encapsulated if the incoming packet is IPv4 packet.

When all MPLS labels are removed exposing IPv4 packet, the packet header is re-aligned to the 5th LW in the memory cache. The cache with exposed IPv4 header can then be passed to IPv4 block for further processing.

4Gb Ethernet IPv6/IPv4 Application 7

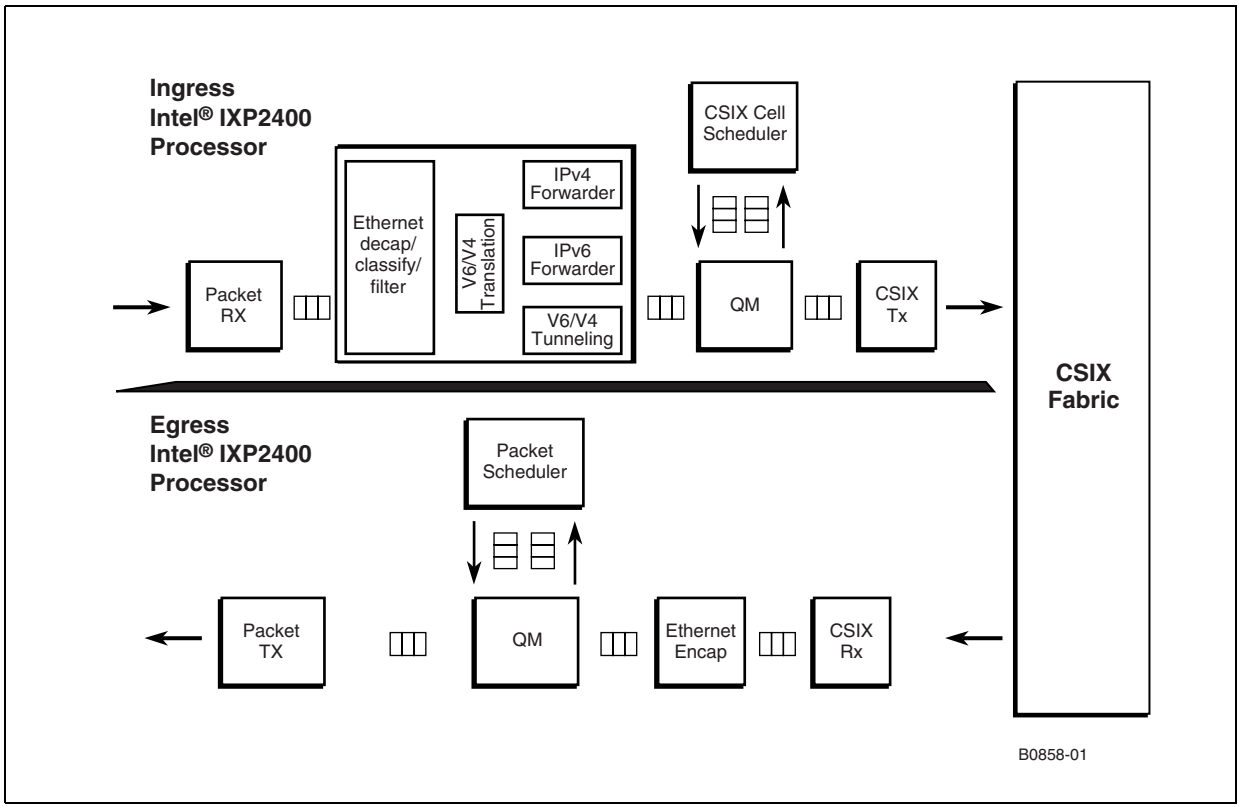
This chapter describes an IPv4 and IPv6 forwarding application for Ethernet implemented on two Intel® IXP2400 Network Processors. The chapter also provides a high-level design overview and lists the different software components used to build this application.

The application described in this chapter is supported on the Intel® IXDP2400 Advanced Development Platform.

7.1 Software Overview

Figure 7-1 shows the software components needed to implement an IPv4 and IPv6 forwarding application for Ethernet. All the context pipe-stages (e.g. Packet RX, Queue Manager, Scheduler etc.) occupy an entire microengine. Each context pipe-stage is mapped to a single microblock running on a ME with or without a dispatch loop. The functional pipeline runs on four microengines, implements the layer-2 (Ethernet) decapsulation, the IPv4 forwarder, IPv6 forwarder, V6/V4 tunneling and translation blocks. The tunneling microblock is required when IPv6 packets need to be tunneled over an IPv4 network. The translation block is required when an IPv4 only host needs to communicate with an IPv6 only host or vice versa.

Figure 7-1. Software Components for IPv4/IPv6 Forwarding and IPv6/IPv4 Tunneling



7.2 Data Flow for the Ingress IXP2400

The following sections describe the data flow on the ingress IXP2400:

7.2.1 Packet RX

This block is identical to the Section 2.2.1.1, “Packet RX” on page 25 except that it sets the header type field in the packet meta data to Ethernet.

7.2.2 Ethernet Decapsulation/Classify/Filter

The Ethernet decapsulation/classify/filter microblock runs in a functional pipeline with the IPv4 microblock, the IPv6 microblock and the IPv6/IPv4 tunneling microblock on four microengines or 32 threads.

This microblock removes the layer-2 Ethernet header from the packet by updating the offset and size fields in the packet metadata. It also implements MAC filtering based on the destination MAC address in the Ethernet header. Based on this filtering, the packet may be dropped.

This microblock also classifies the packet into IPv4, IPv6, MPLS, ARP etc. If the packet is an ARP packet, it is marked as an exception packet to be sent to the Intel XScale® core (`IX_EXCEPTION`). Otherwise the packet is sent down the microengine pipeline for further processing. In this application, the dispatch loop silently drops packets classified as MPLS.

7.2.3 V6/V4 Translation Microblock

The Translation microblock implements an IPv6-IPv4 translation mechanism. The translation mechanisms allow IPv6 and IPv4 hosts to coexist, which allows an IPv6 host and IPv4 host to communicate with each other. These mechanisms are designed to support the scenario/case where an IPv6-only network may be deployed, but there is a need (OR the machines in the network) need to gain access to the resources in an IPv4-only network. The translation microblock provided supports the NAT-PT translation mechanism. The primary function of the translation microblock is to change the IP headers in the relevant packets as they pass through, which to each of the endpoints appears as if they are talking to a host with the same network layer.

The translation microblock examines the source or destination IP addresses in each packet to determine if the addresses need translation. If an appropriate packet is identified, the microblock extracts information from the existing IP header and converts it to the desired format. The translation microblock also updates the packet meta-data so that the downstream microblocks can work with the packet as if it had arrived in translated format. If a packet need not be translated the microblock simply passes it on.

7.2.4 IPv4 Forwarder

This block is identical to the block described in Section 2.2.1.3, “IPv4 Forwarder” on page 26.

7.2.5 IPv6 Forwarder

IPv6 is a new version of the internet protocol, designed as the successor of IPv4. IPv6 addresses are 128 bits long, which solves the “address exhaustion” problem that IPv4 is facing. Besides the expanded addressing capabilities of IPv6, the changes from IPv4 to IPv6 include header format simplification, improved support for extensions and options (the basic header size is now fixed, which makes processing common-case packets really simple and fast), flow labelling capability (provides flow labels to support “real-time” traffic) and support for authentication and privacy capabilities.

The IPv6 forwarder microblock validates the IP header per RFC 2460. If the validity checks fail, then the packet is set up to be dropped as specified in [IXASF]. Otherwise a Longest Prefix Match (LPM) is performed on the IPv6 header. The result is an IPv6 next-hop ID, a fabric blade id (identifying a unique IXP2400 on the fabric) and an output port identifying the output port on the egress IXP2400. The next-hop ID is passed over the CSIX fabric to an Egress IXP2400 where it is used to look up information about the Layer-2 header to be prepended to the packet buffer. The output port is also passed over the CSIX fabric to the egress IXP2400 and is used to transmit over the appropriate port. All three fields are stored in the packet metadata.

If no match is found, then the packet is set up to be sent up to the XScale core for further processing as specified in [IXASF]. Packets are also sent to the core in a number of other cases, for example when the packet is destined for a local interface or is to be fragmented.

From the IPv6 forwarder block, the packet is passed on to an application specific system microblock (DL_QM_Sink[]). This microblock checks if the packet is to be dropped or sent to the XScale Core. If not, it sends an enqueue request to the Queue Manager over a scratch ring. The DL_QM_Sink[] also writes the cached packet header to DRAM and the packet meta information to SRAM.

7.2.6 IPv6/IPv4 Tunneling Microblock

Tunneling of IPv6 packets in IPv4 packets is used in several transition mechanisms that allow the coexistence of both IPv6 and IPv4 on a network. Tunneling supports communication between two IPv6 “islands” connected by an IPv4 “cloud”. This scenario exists today because only some parts of the Internet have made the transition to IPv6. The rest of the Internet is still IPv4 based. Put simply, *tunneling* encapsulates IPv6 packets within IPv4 packets, which are sent over an IPv4 network. When the packet reaches the tunnel “end-point” the IPv4 header is stripped and the IPv6 packet is delivered to the destination node. The tunneling microblock performs the encapsulation and decapsulation functionality.

The tunneling microblocks are assumed to be run as part of a packet forwarding microblock group that includes both the IPv4 and IPv6 forwarders. After the L2 header is removed and the packet is classified either as an IPv4 or IPv6 packet, the packet metadata is updated to point to the L3 header. The IPv4 forwarder performs any required header validation, and performs a lookup based on the IPv4 destination address. The lookup sets the next-hop identifier in the metadata cache. The IPv4 forwarder reads a portion of the next-hop information and determines which microblock must execute next. If the next-hop information indicates that the destination address does not represent a tunnel endpoint, the packet is passed on to the next stage for L2 header processing. If the next-hop information indicates that the destination address represents a tunnel endpoint, the packet is passed on to the V6V4-Tunnel-Decap microblock. The V6V4-Tunnel-Decap microblock validates the source address if necessary, removes the IPv4 header and passes the IPv6 packet on to the IPv6 forwarder.

The IPv6 forwarder performs any required header validation and performs a route lookup based on the destination address of the packet. The lookup sets the next-hop identifier in the metadata cache. The forwarder then reads a portion of the next-hop information to determine which microblock must execute next. If the next-hop information indicates that the next-hop does not require a tunnel, the packet is passed on to the L2 processing stage. If the next-hop information indicates that the destination is reachable via a V6 over V4 tunnel, the IPv6 forwarder passes the packet to the V6V4-Tunnel-Encap microblock. The V6V4-Tunnel-Encap microblock encapsulates the packet with an IPv4 header and passes the packet to the IPv4 forwarder. The IPv4 forwarder then performs a lookup and sets the next-hop identifier as described earlier. The packet if finally passed on to the L2 processing stage.

7.2.7 Cell Based Queue Manager (Cell QM)

This block is identical to the block described in Section 2.2.1.4, “Cell Based Queue Manager (Cell QM)” on page 26.

7.2.8 CSIX Scheduler

This block is identical to the block described in Section 2.2.1.5, “CSIX Scheduler” on page 27.

7.2.9 CSIX TX

This block is identical to the block described in Section 2.2.1.6, “CSIX TX” on page 27.

7.3 Data Flow for the Egress IXP2400

This section describes the data flow for the Egress IXP2400.

7.3.1 CSIX RX

This block is identical to the block described in Section 2.2.2.1, “CSIX RX” on page 28.

7.3.2 Ethernet Encapsulation

This block is identical to the block described in Section 3.2.2.2, “Ethernet Encapsulation” on page 44

7.3.3 Packet Based Queue Manager (Packet QM)

This block is identical to the block described in Section 2.2.2.3, “Packet Based Queue Manager” on page 28.

7.3.4 Egress Scheduler

This block is identical to the block described in Section 2.2.2.4, “Egress Packet WRR/DRR Scheduler” on page 28.

7.3.5 Packet TX

This block is identical to the block described in Section 2.2.2.5, “Packet TX” on page 29.

7.4 Dispatch Loops / Microblock Groups

There are two dispatch loops (microblock groups) on the ingress pipeline

- Dispatch Loop for the Packet Frame Reassembly Stage (Figure 7-2)
- Dispatch Loop for the IPv4 forwarder, IPv6 forwarder and V6/V4 tunneling functional pipeline (Figure 7-3)

The QM, Scheduler and CSIX TX blocks don't use a dispatch loop (they still use the dispatch loop macros where required).

Figure 7-2. Dispatch Loop for the Packet Frame Reassembly Stage

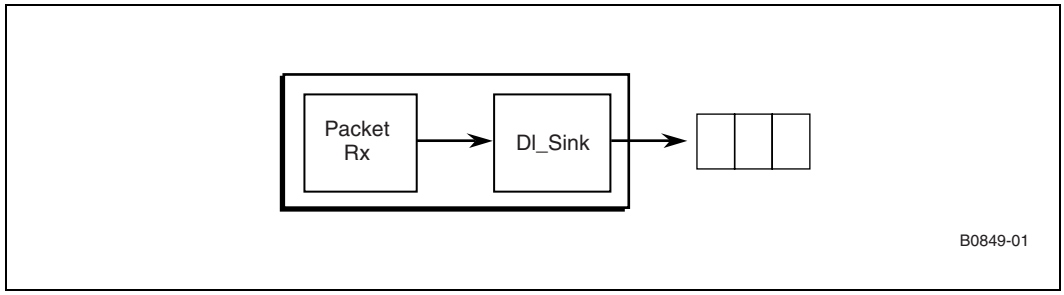
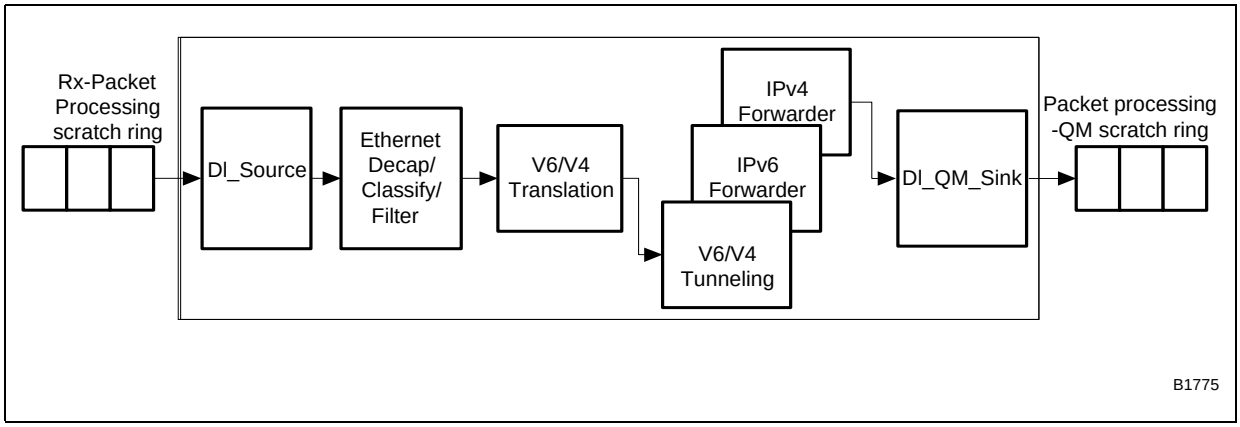


Figure 7-3. Dispatch Loop for the IPv4, IPv6 and V6/V4 Tunneling Functional Pipeline



Note that the system microblocks `dl_source, dl_sink, dl_qm_sink,` and so on are application specific. They may be changed for different packet processing pipelines.

There are two dispatch loops (microblock groups) on the egress pipeline

- Dispatch Loop for the CSIX RX Reassembly stage (Figure 7-4)
- Dispatch Loop for the Ethernet encapsulation stage (Figure 7-5)

Figure 7-4. Dispatch Loop for CSIX Reassembly Stage

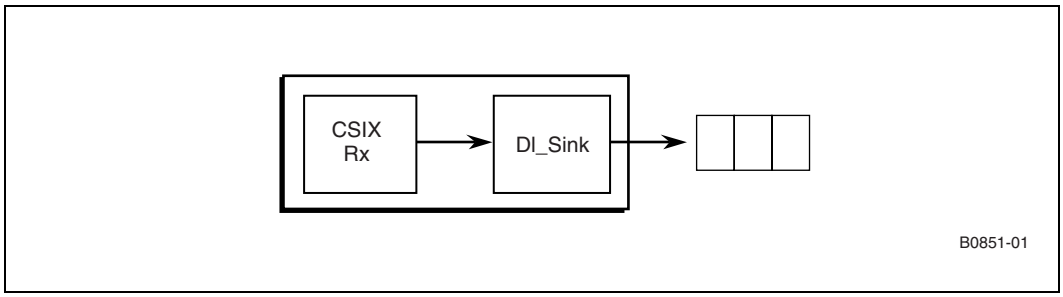
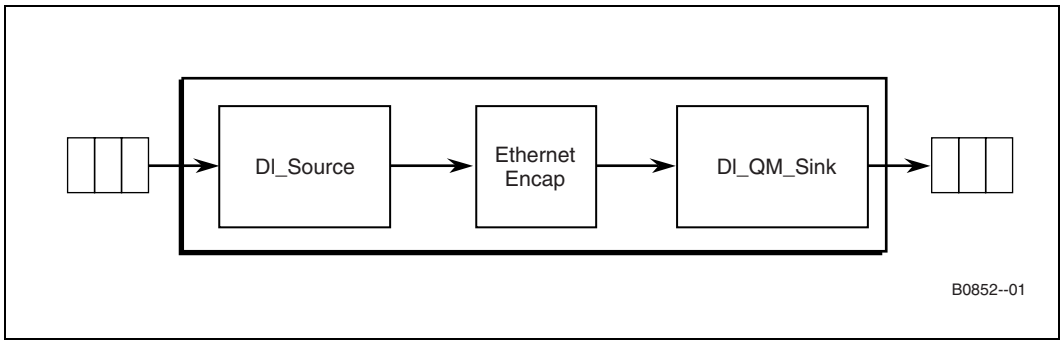


Figure 7-5. Dispatch Loop for Ethernet Encapsulation Stage



7.5 Performance Analysis

The IXP2400 operates at 600 MHz. For a min Ethernet packet of 78B, the packet inter-arrival time at 4 Gbps line rate is 117 ME cycles. In order to maintain line rate for min packets, each stage of the pipeline cannot exceed this budget. In other words, each stage of the pipeline needs to retire a packet every 117 cycles. Table 7-1 summarizes the performance analysis for the IPv6 Ethernet pipeline.

Table 7-1. Performance Analysis for the IPv6 Ethernet Pipeline

Line rate for 4 Gig ports	4 Gigabits/sec
Min Ethernet packet size	78 bytes (+ 20 byte inter packet gap)
Packet Throughput for min packets	5.10 million packets/sec = (4 / (98*8)) * (10**9)
IXP2400 clock frequency	600 MHZ
Inter-packet arrival time for min packets	600/5.10 = 117.64 cycles
Compute cycles per packet for a context pipe stage	117
Latency per packet for a context pipe stage	117 * 8
Compute cycles per packet for a functional pipeline of n microengines	117*n
Latency per packet for a functional pipeline of n microengines	117*8*n

DiffServ for POS Application 8

This chapter describes a DiffServ application for Packet over SONET (POS) implemented on two half-duplex Intel® IXP2400 Network Processors connected to a CSIX switch fabric. It provides a high-level design overview and lists the different software components used to build the application.

The application described in this chapter is supported on the Intel® IXDP2400 Advanced Development Platform.

8.1 Hardware Overview

This release of DiffServ blocks runs on the Intel® IXDP2400 Advanced Development Platform. The platform is comprised of a chassis with the following cards:

- Intel® IXDP2400 Advanced Development Platform Base Card: A 9U baseboard with dual IXP2400 Network Processors and a switch fabric connector with loop back
- Intel® IXD2448 Single OC-48 I/O Option Card: A PoS OC-48 modular media card for the base card
- A passive Switch Fabric loopback card for the base card

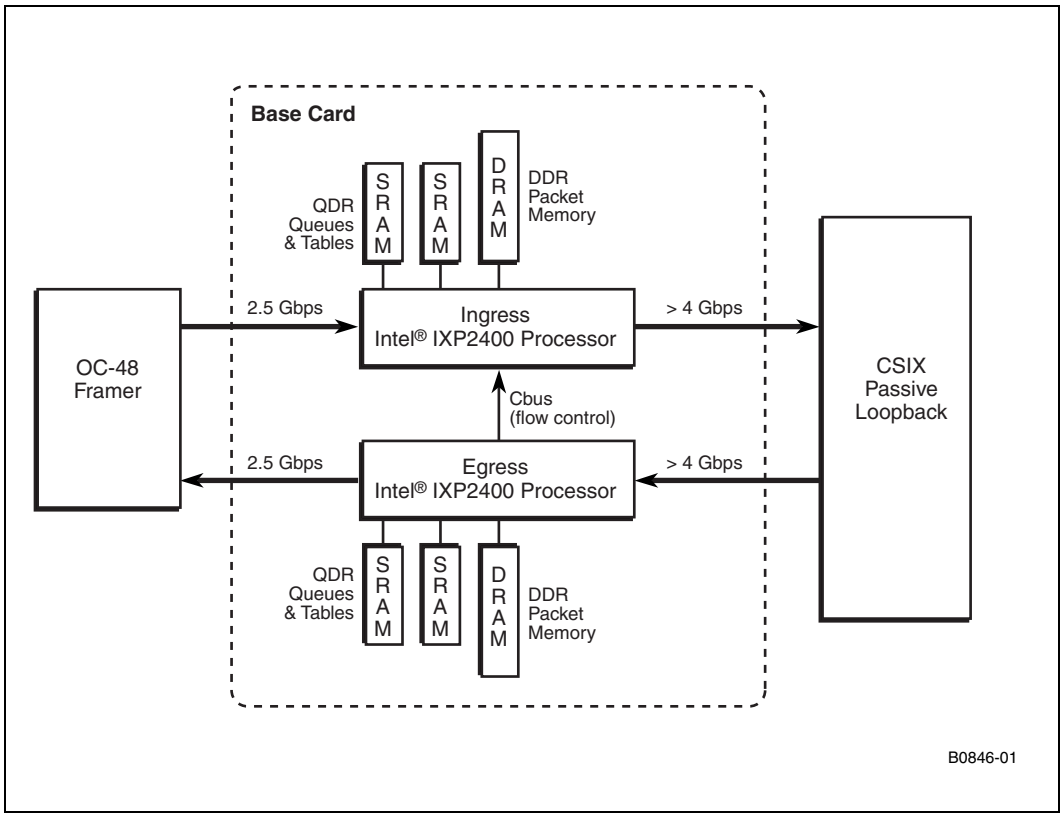
In future releases, the platform can be extended with TCAM chips to speed up classification.

As illustrated in Figure 8-1, the development platform contains two network processors, dedicated to ingress and egress packet processing, respectively.

The ingress processor receives data from the OC-48 interface. It reads POS frames, assembles them into packets and removes L2 headers. Next, it classifies packets, polices them and makes a forwarding decision. Finally, the ingress processor transmits packets, along with results of the ingress classification, towards the CSIX fabric. Prior to transmission, packets are segmented into CSIX frames. In this release, the fabric is reduced to a passive loopback between the ingress to egress network processors.

The egress processor reassembles CSIX frames back into IP packets, applies the required QoS service, and transmits packets over the POS interface.

Figure 8-1. IXDP2400 Advanced Development Platform Overview



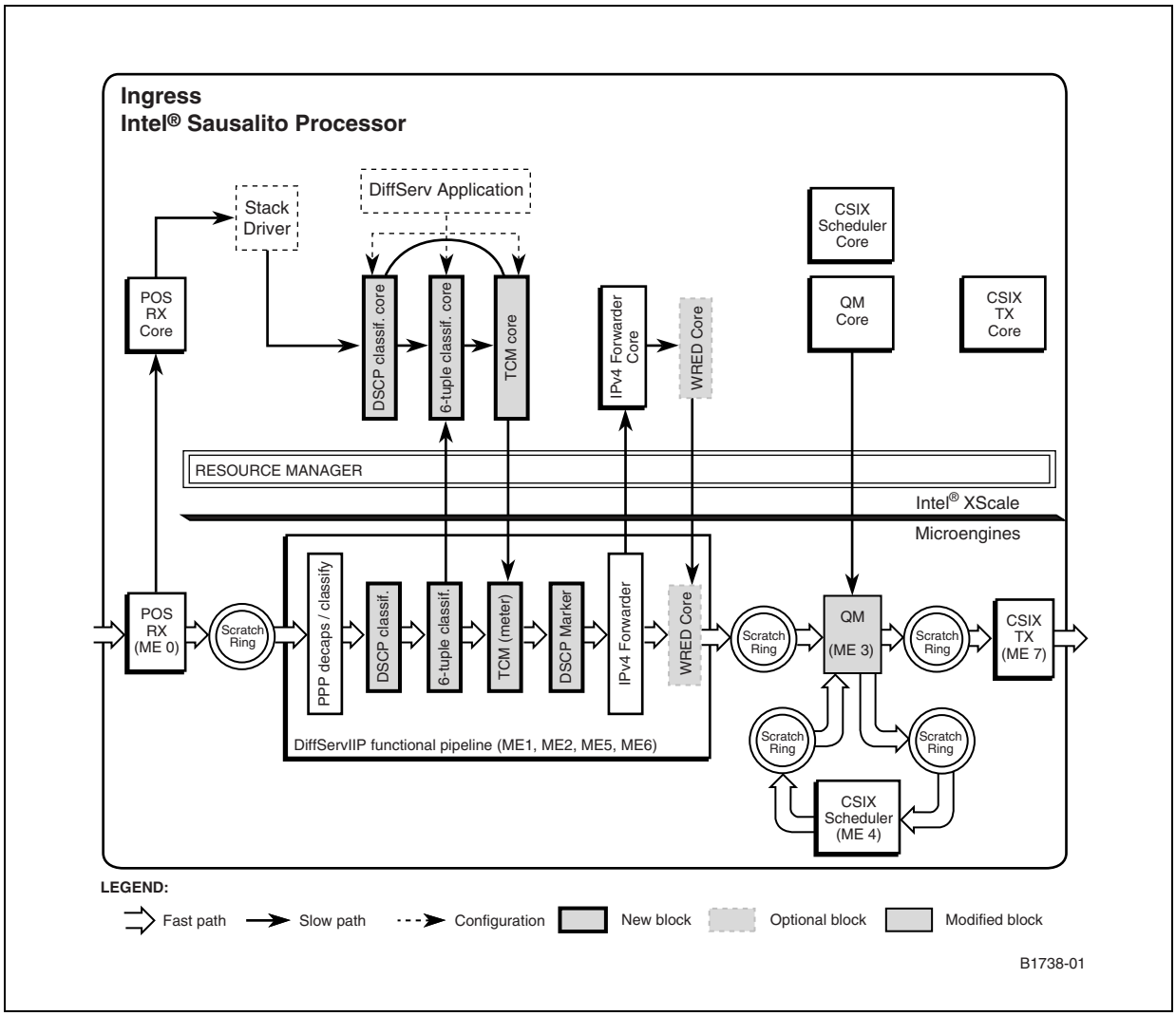
8.2 Software Overview

8.2.1 Ingress IXP2400 Network Processor - DiffServ/IPv4

Figure 8-2 details the software architecture of DiffServ/IPv4 blocks on the ingress processor. The diagram shows mapping of functional blocks to microengines. The physical assignment determines inter-microengine communications such as scratch rings or next neighbor registers. The arrangement affects also utilization of the internal Command bus and S-Push/Pull buses.

The DiffServ blocks (shaded boxes) extend the IPv4 POS application (white boxes) described in Chapter 2, “OC-48 POS IPv4 Forwarding Application.”

Figure 8-2. IPv4 and DiffServ—Ingress Blocks



8.2.1.1 Packet RX Microblock

The Packet RX block runs as a context pipeline on one microengine. It reassembles PPP packets coming form the OC-48 media interface. The Packet RX microengine uses a scratch ring for communication with next blocks. This block is not modified for a DiffServ application. All messages posted in scratch/NN rings have the same format as described in Chapter 4, “Packet RX Microblock.”

8.2.1.2 DiffServ/IPv4 Functional Pipeline

The DiffServ/IPv4 functional pipe executes in parallel on four microengines. The pipeline is organized in a dispatch loop which starts with a 6-tuple classification and metering. In this way, packets dropped by a meter do not go through IP lookup. Alternatively, the SRTCM/DSCP blocks can be moved after the IPv4 forwarder. In such case, packets with invalid headers—for example, TTL expired—won't get unnecessarily metered.

Both arrangements give the same performance in the worst case when all packets have valid headers and are always marked. The ingress dispatch loop can optionally contain WRED congestion avoidance (not shown in Table 8-2).

Note: The ingress-side WRED accommodates multi-blade environments and will not be implemented on the IXDP2400 Advanced Development Platform.

8.2.1.2.1 PPP Decapsulation /Classify Microblock

The PPP decapsulation/classify microblock removes the layer-2 PPP header from the packet. It also classifies the packet into IPv4, IPv6, PPP control packet (LCP, IPCP etc.). PPP control packets are thrown to the XScale core component. IPv4 packets are passed on to the 6-tuple classifier for further processing. IPv6 packets are dropped in this release.

8.2.1.2.2 6-tuple Classifier Microblock

The 6-tuple classifier microblock performs an exact-match lookup on the IPv4 header. The classifier maintains a hash table with statically configured exact-match rules. Thus, a lookup can fail only if there is no static rule defined. An empty rule corresponds to best-effort traffic. As a result, on lookup failure a packet is assigned to the best-effort service (default rule) and passed on for further processing. The classifier core component configures a hash table used by microblock. In addition, it handles packets generated by a local TCP/IP stack and exception packets with IP header options passed by the classifier microblock.

8.2.1.2.3 TCM Meter Microblock

The TC meter implements two metering algorithms: Single Rate Three Color Meter (SRTCM) described in (www.ietf.org/rfc/rfc2697.txt) and Two Rate Three Color Meter (TRTCM) decribed in (www.ietf.org/rfc/rfc2698.txt). The algorithms contain a critical section (read-modify-write operation). For that reason, only the microblock processes packets, while the core component deals with configuration issues. If the core processed packets, it would have to synchronize its operations with microengines, in order to avoid corruption of shared data structures. To decrease design complexity, it is assumed that Xscale core components do not execute critical sections at all.

8.2.1.2.4 DSCP Marker Microblock

The DSPC marker updates TOS field in the IP header. This operation does not result in exception packets, nor it requires a critical section.

8.2.1.2.5 IPv4 Forwarder Microblock

The IPv4 Forwarder microblock validates the IP header as per RFC 1812. Invalid packets are dropped. Otherwise, a microblock performs Longest Prefix Match (LPM) on the IP destination address. The lookup result specifies destination where a packet should be forwarded. This block is not modified for a DiffServ application.

Ideally, IP header sanity checks shall be done prior to 6-tuple classification. Unfortunately, the existing IPv4 combines header validation procedure with the LPM algorithm. Thus, a design decision is to leave header validation after 6-tuple classification.

8.2.1.2.6 WRED Microblock

Doing WRED on ingress is optional, since the main aim of RED is to prevent persistent (long-term) queues. The IXDP2400 Advanced Development Platform simulates switching fabric with a hardware loopback. Thus, ingress queuing can have transient character only, if any. For that reason in the IXDP2400 Advanced Development Platform, the ingress processor does not include WRED.

8.2.1.3 Ingress Queue Manager for DiffServ

The ingress Queue Manager performs enqueue/dequeue operations on the hardware-assisted SRAM queues. The Queue Manager receives enqueue requests from the IPv4/DiffServ pipeline through a scratch ring. Another scratch ring is fed with dequeue requests from the CSIX scheduler. When the queue state changes between empty and non-empty, Queue Manager sends a transition message to the Scheduler (via Next Neighbor registers). After every dequeue operation, the QM passes a transmit request to the scratch ring served by the TX microblock. All messages posted in scratch/NN rings have the same format as described in Chapter 2, “System Data Structures and Design Choices.”

The only modification is that when WRED block is used, the ingress Queue Manager flushes queue lengths and last idle timestamp to SRAM memory.

Note: In this release, the Ingress Queue Manager for DiffServ is a separate microblock from the Cell-Based Queue Manager described in Chapter 13, “Queue Manager For OC-48 Microblock.”

8.2.1.4 CSIX Scheduler

This CSIX scheduler selects constant-length packet segments (cframes) to be transmitted to the CSIX fabric. The scheduler employs Round Robin (RR) among the fabric ports and Weighted Round Robin (WRR) among the port queues. The scheduler handles also flow control messages received from the fabric. This microblock is the same as the one used for IPv4 POS, as described in Chapter 17, “Fabric Scheduler For OC-48.”

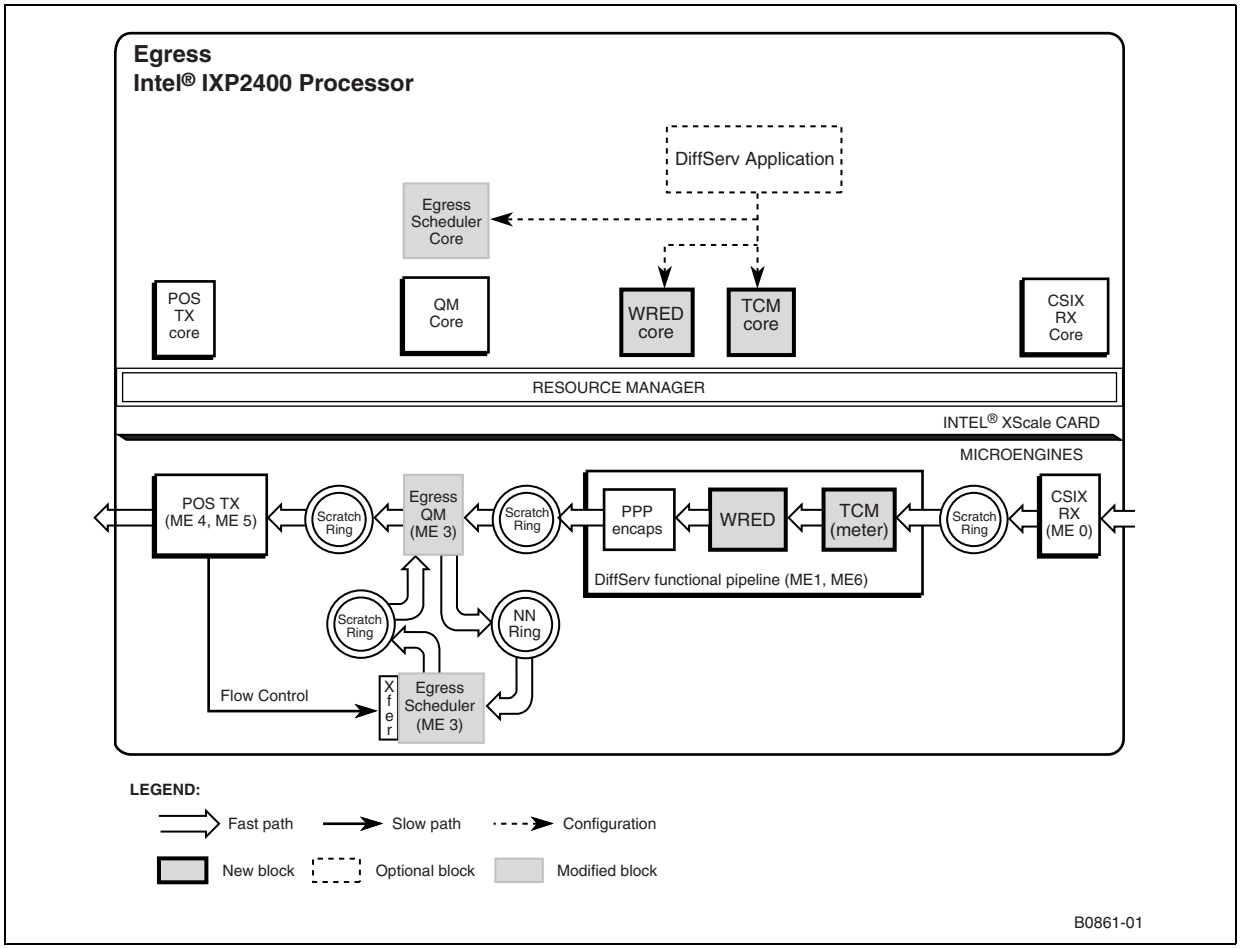
8.2.1.5 CSIX TX Microblock

The CSIX TX microblock receives transmit messages from the Queue Manager and moves packet segments (cframes) into a transmit buffer. It also encapsulates cframe payload with a CSIX header, and a proprietary Traffic Manager (TM) header. The CSIX/TM headers convey metadata information to the egress processor. This microblock is the same as the one used for IPv4 POS, as described in Chapter 7, “CSIX TX Microblock.”

8.2.2 Egress IXP2400 Network Processor—DiffServ/ IPv4

Figure 8-3 details the software architecture of DiffServ/IPv4 blocks on the egress processor. The diagram shows mapping of functional blocks to microengines.

Figure 8-3. IPv4 and DiffServ: Egress Architecture



8.2.2.1 CSIX RX Microblock

The CSIX RX block reassembles cframe segments back into packets, and restores metadata information. Next, it passes a packet to the egress DiffServ blocks, using a scratch ring for communication.

8.2.2.2 DiffServ Functional Pipeline

The next two microengines run an egress-side DiffServ functional pipe stage. The pipeline is a part of Per Hop Behaviors implementation. The “canonical” EF PHB implementation comprises a priority queue protected with a token bucket rate-limiter—refer to “An Expedited Forwarding PHB” (www.ietf.org/rfc/rfc3246.txt). The rate limiter can be satisfied with the properly configured SRTCM block. The Assured Forwarding PHB starts with WRED congestion avoidance, followed

by a DRR scheduler. Both SRTCM and WRED algorithms implement a critical section. For that reason, only the microblocks process packets. The core component functionality is limited to configuration and management procedures.

The PPP encapsulation microblock adds the layer-2 PPP header to the packet and enqueues it to the next stage of the pipeline.

8.2.2.3 Egress Queue Manager

The egress Queue Manager block is virtually identical to its ingress counterpart, except that it dequeues packets not segments. Moreover, this block returns dequeue response messages to the scheduler. The response contains length of a dequeued packet, which is needed by a DRR algorithm. Additionally, it generates enqueue/dequeue message—for each packet and not just upon queue transitions between empty and non-empty states.

8.2.2.4 Egress Scheduler

The Egress Scheduler schedules POS packets to be transmitted over the POS interface. It implements Weighted Round Robin (WRR) scheduling among the ports, Strict Priority (SP) between two sets of port queues, and Deficit Round Robin (DRR) scheduling among the queues belonging to one priority group.

8.2.2.5 Packet TX Microblock

The Packet TX microblock transmits packets over the POS interface. It moves the packets to the transmission buffers of 16 virtual output ports.

8.2.3 Performance Analysis

The analysis is same as for the plain IPv4/POS application—refer to the Section 2.3, “Performance Characterization” on page 31. In brief, the IXP2400 operates at 600 MHz. For a min POS packet of 49B, the packet inter-arrival time at OC-48 line rate is 97 microengine cycles. In order to maintain line rate for min packets, each stage of the pipeline cannot exceed this budget.

8.3 System Data Structures and Resource Allocation

This section describes system-wide data structures used by DiffServ application. It also describes how system resources—for example, microengines, scratch rings, NN rings, memory regions, and others—are allocated and used among the different microblocks. This chapter focuses on DiffServ blocks; details on plain IPv4/POS structures can be found in the Chapter 2, “System Data Structures and Design Choices”.

8.3.1 Ingress System Resource Allocation

The allocation of ingress microengines is same as in the plain IPv4/POS application—refer to the Section 2.4, "Ingress System Resource Allocation" on page 32. Table 8-1 shows memory regions added by DiffServ microblocks. For performance reasons, all DiffServ structures are placed in SRAM. However, in a cost-oriented application it is recommended to put hash table in DRAM.

Table 8-1. Ingress IXP2400 Memory Usage

Item	Size per entry (in bytes)	Number of entries	Total SRAM used	Total DRAM used	Total scratch used
Plain IPv4/POS application	-	-	9.15MB	64MB	10kB
6-tuple classifier hash table	32	64k	2 MB		
6-tuple classifier collision chains	32	32k	1 MB		
6-tuple classifier 64-bit stats	16	96k	1,5 MB		
TCM table	64	1k	64 kB		
TCM 64-bit stats.	32	1k	32 kB		
DSCP classifier table	8	16k	128 kB		
DSCP classifier 64-bit stats.	16	16k	256 kB		
Total			**14.52 MB**	**64 MB**	**10 kB**

Note: The hash table size can be much smaller in an OC-48 reference application. This is because the flow-cache model (with dynamic hash entries) does not scale to high-speed links. Thus, only statically configured hash entries are supported, and it is not likely that one configures all 64k of rules.

8.3.2 Egress System Resource Allocation

On the egress IXP2400 Network Processor, one microengine is added to accommodate DiffServ PHBs, as compared with plain IPv4/POS application. Table 8-2 shows the modified microengine allocation.

Table 8-2. Egress IXP2400 Microengine Allocation

Microblock	ME#	Communication with previous block
CSIX RX	ME0	Auto-push status from MSF
SRTCM + WRED + PPP encapsulation	ME1, ME6	Scratch Ring
Egress QM	ME 2	Scratch Ring
Egress Scheduler	ME 3	Next neighbor + xfer reflector registers
Packet TX	ME4, ME5 (MPHY-16)	Scratch ring
Unused (available headroom)	ME7	

Table 8-3 shows memory regions added by DiffServ microblocks on the egress processor.

Table 8-3. Egress IXP2400 Memory Usage

Item	Size per entry (in bytes)	Number of entries	Total SRAM used	Total DRAM used	Total scratch used
Plain IPv4/POS application	-	-	1.04 MB	64MB	10kB
Queue Descriptors entry extension	16	1024	16 kB		
SRTCM meter table	64	256	16 kB		
SRTCM 64-bit stats.	32	256	8 kB		
WRED table	64	256	16 kB		
WRED 64-bit stats.	32	256	8 kB		
Total			**1.11 MB**	**64 MB**	**10 kB**

8.3.3 Buffer Handle

A network processor stores packets in fixed-size buffers, chaining them if needed for large packets. Every buffer consists of a buffer data area in DRAM and a packet metadata in SRAM (there is 1-1 mapping between DRAM buffers and SRAM metadata). For DiffServ application, the data portion of a DRAM buffer is 2048 bytes. This size is configurable as long as it is set to a power of two. The metadata size is 32 bytes (see Section 8.3.4, “Packet Metadata” on page 109).

A 32 bit long buffer handle uniquely identifies both buffer data area and packet metadata. See Chapter 2, “System Data Structures and Design Choices” for details.

8.3.4 Packet Metadata

Chapter 2, “System Data Structures and Design Choices” describes the generic layout of a packet metadata. The first 8 bytes (2 long words) are always the same for every application. The remaining fields depend on an application. In a DiffServ scenario, the metadata structure is almost the same as defined in Section 2.2, “Packet Meta Data (Buffer Descriptor)” on page 58. The only new field is `color_id` used by SRTCM and WRED blocks. Table 8-4 Packet Metadata structure

Table 8-4. Packet Metadata Structure

LW	Bits	Size	Field	Description
0	31:0	32	buffer_next	Buffer handle of next buffer in the packet chain
1	31:16	16	buffer_size	Buffer size in bytes
	15:0	16	offset	Offset of the start of data in the buffer in bytes
2	31:28	16	packet_size	Total packet size across all chained buffers
	15:12	4	free_list_id	Free list ID for buffer
	11:8	4	rx_stat	Receive Status Flag
	7:0	8	header_type	Type of header at offset bytes into the packet
3	31:16	16	input_port	Input port on ingress processor
	15:0	16	output_port	Output port on egress processor

Table 8-4. Packet Metadata Structure (Continued)

LW	Bits	Size	Field	Description
4	31:16	16	next_hop_id	Identifier of a next hop IP node
	15:8	8	fabric_port	Output port for fabric indicating a destination blade
	7:4	4	reserved	Currently not used
	3:0	4	nexthop_id_type	ID specifying in which table to lookup the `next_hop_id`
5	31:0	32	flow_id	Flow id (QoS flow id or MPLS label/flow id)
6	31:16	16	class_id	Relative identifier of a queue within an output port
	1:0	2	color_id	Packet drop precedence level (green, yellow, red)
	15:2	14	reserved	Currently not used
7	31:0	32	packet_next	Pointer to next packet (unused in cell mode)

The metadata for the first buffer in a packet chain contains all these fields. For the remaining buffers, only the first two long words are relevant. The rest are not used.

8.4 Interfaces Between the Various Microblocks

8.4.1 Inter-Microengine Messages

This section describes the interfaces between microengines on ingress and egress IXP processors for a DiffServ application. The interfaces are described in terms of messages exchanged over scratch and NN rings. To ensure backward compatibility and easy migration, most of these interfaces are unchanged as compared with the IPv4 reference design described in Section 2.6, "Interfaces Between the Various Microblocks" on page 34. This section highlights only modifications.

8.4.1.1 POS RX and Ingress DiffServ/IPv4 Functional Pipeline

Not changed—see Section 2.6.1, "Packet RX and Packet Processing Microengines" on page 35.

8.4.1.2 Ingress DiffServ/IPv4 Functional Pipeline and Ingress Queue Manager

Not changed—see Section 2.6.2, "Packet Processing Microengines and Cell Queue Manager" on page 35.

8.4.1.3 Ingress Queue Manager and Ingress Scheduler

Not changed—see Section 2.6.3, "Cell Queue Manager and CSIX Scheduler" on page 36.

8.4.1.4 Ingress Queue Manager and CSIX TX

Not changed—see Section 2.6.4, “Cell Queue Manager and CSIX TX” on page 36.

8.4.1.5 CSIX RX and Egress DiffServ Pipeline

The interface between the CSIX RX pipe-stage and the egress DiffServ functional pipeline is a scratch ring. Table 8-5 shows each entry in the scratch ring, which is 4 long words and the message format between CSIX RX and egress DiffServ pipeline.

Table 8-5. Message Format Between CSIX RX and Egress DiffServ Pipeline

LW	Bits	Size	Field	Description
0	31:0	32	dl_buffer_handle	Buffer Handle for the SOP Descriptor
1	31:0	32	dl_eop_buffer_handle	Buffer Handle for EOP Descriptor (may be NULL)
2	31	1	valid_bit	Must be 1
	30:26	5	reserved	Not used
	25:24	2	color_id	Packet drop precedence level
	23:16	8	reserved	Not used
	15:8	8	packet_size	Total packet size across buffers
	7:4	4	output_port	Output port on an egress blade
	3:0	4	class_id	Relative identifier of an output queue on the output port.

8.4.1.6 Egress DiffServ Pipeline and Egress Queue Manager

Same as interface between ingress DiffServ/IPV4 functional pipeline and the Ingress Queue Manager. See Section 2.6.6, “PPP Encap and Packet Queue Manager” on page 37 for details.

8.4.1.7 Egress Queue Manager and Scheduler

Table 8-6 shows the NN Ring Message format between egress Queue Manager and packet Scheduler.

Table 8-6. Message Format Between Egress Queue Manager and Scheduler

LW	Bits	Size	Field	Description
0	31	1	reserved	Not used
	30	1	enqueue_event	If set to 1, one packet has been enqueued.
	29:16	14	reserved	Not used
	15:0	16	enq_queue_id	Absolute queue identifier for enqueue event (`queue_id = port_id*QUEUES_PER_PORT + class_id`)

Table 8-6. Message Format Between Egress Queue Manager and Scheduler (Continued)

LW	Bits	Size	Field	Description
1	31	1	reserved	Not used
	30	1	dequeue_event	If set to 1, one packet has been dequeued.
	29:24	6	reserved	Not used
	23:16	8	packet_size	Packet size in 128-byte chunks
	15:0	16	deq_queue_id	Absolute queue identifier for dequeue event (`queue_id = port_id*QUEUES_PER_PORT + class_id`)

8.4.1.8 Egress Queue Manager and POS TX

See Section 2.6.8, "Packet Queue Manager and Packet TX" on page 38

8.4.2 Ingress Dispatch Loop Variables

Microblocks running on the same microengine constitute a functional pipeline organized in a dispatch loop. It is not desirable that every block in a loop reads packet metadata variables and writes it back to SRAM. Instead, at the beginning a dedicated block (`Dl_source`) copies required metadata fields to global state variables, which may be cached in registers or local memory. At the loop end, another block (`Dl_sink`) writes back the modified fields to SRAM if necessary. All other blocks operate on dispatch loop variables.

A set of cached variables depends on an application scenario. Table 4 9 lists dispatch loop variables relevant for DiffServ blocks in ingress pipeline. Most variables are the same as described in [IXA DM], except for `color_id`.

Table 8-7. Ingress Dispatch Loop Variables for DiffServ Application

Field Name	Bits	Description
exception_id	8	Microblocks use this variable when sending packets to the Intel® XScale™ core. The exception_id should be set to identifier of a microblock, which generates an exception.
exception_code	8	The microblock sets an 8-bit exception code when a packet is sent to the Intel® XScale™ core component. The code is opaque to dispatch loop and Resource Manager.
dl_next_block	8	Identifier of a next microblock to continue with packet processing.
dl_buf_handle	32	The handle for the buffer containing the start of the packet.
dl_packet_size	16	The total length of the packet across multiple buffers.
dl_input_port	16	The logical port number on which the packet was received on an ingress blade. An existing POS application supports 16 ports.
dl_output_port	16	The logical port number where a packet is to be transmitted on an egress blade. An existing POS application supports 16 ports.
dl_fabric_port	8	Identifier of an egress blade (used when multiple blades are connected to the switching fabric).
dl_header_type	4	The type of packet stored at "offset" bytes in a DRAM buffer.
dl_next_hop_id	16	The IP next hop identifier, pointing to forwarding information.

Table 8-7. Ingress Dispatch Loop Variables for DiffServ Application (Continued)

Field Name	Bits	Description
dl_nexthop_id_type	4	The type of table where the next_hop_id is defined.
dl_flow_id	32	The flow identifier used for packet metering and policing.
dl_class_id	16	The relative identifier of an output queue, within an output port. This is set when classifying packets for QoS processing.
dl_color_id	2	The packet dropping precedence level, often referred to as green, yellow or red color.

A microblock never accesses these variables directly. Instead, it uses a set of helper functions, which are provided as a part of IXA framework.

8.4.3 Egress Dispatch Loop Variables

On egress, the functional pipeline comprises SRTCM meter and WRED congestion avoidance. These blocks implement DiffServ Per Hop Behaviors. Table 8-8 lists dispatch loop variables relevant for these blocks.

Table 8-8. Egress Dispatch Loop Variables for DiffServ Application

Field Name	Bits	Description
dl_next_block	8	Identifier of a next microblock to continue with packet processing.
dl_buf_handle	32	The handle for the buffer containing the start of the packet.
dl_packet_size	16	The total length of the packet across multiple buffers.
dl_header_type	4	The type of packet stored at “offset” bytes in a DRAM buffer.
dl_next_hop_id	16	The IP next hop identifier, pointing to forwarding information.
dl_output_port	16	The logical port number where a packet is to be transmitted on an egress blade. An existing POS application supports 16 ports.
dl_flow_id	32	The flow identifier used for packet metering and policing.
dl_class_id	16	The relative identifier of an output queue, within an output port. This is set when classifying packets for QoS processing.
dl_color_id	2	The packet drop precedence level, often referred to as green, yellow or red color.

The egress DL source does not need to retrieve the above variables from a metadata structure in SRAM. It can reconstruct them from a message received over a scratch ring from the CSIX RX microblock. Moreover, the egress `DL_sink` does not need to flush dispatch loop variables back to SRAM, because they are needed no longer.

8.5 Dynamic Behavior

8.5.1 Ingress Data Flow

This section presents data flow between microblocks on an ingress network processor. The data flow is discussed in terms of metadata variables (recall Section 8.2.3, “Performance Analysis” on page 107) transmitted along with a packet. Brackets indicate variables that are set in a former block (right bracket) or consumed by a next block (left bracket).

Table 8-9. IPv4 and Diffserv Ingress Dispatch Loop Variables

Metadata	POS RX + decap	DSCP classif.	6-tuple classif	TCM	IPv4 fwd	WRED	QM + Sched	CSIX TX
packet header	>	<	<		<			
input_port	>	<	<		<			
packet_size	>			<	<	<		
flow_id		>	>	< >				<
class_id		>	>			<	<	<
color_id		>	>	(<) >		<		<
output_port					>			<
fabric_port					>	<	<	<
next_hop_id			>		>			<
next_hop_id_type			>		>			

The Packet RX block first sets up three long words of a metadata structure in SRAM (refer to Table 2-3, “Packet Metadata Format” on page 58). It also puts a buffer handle into a scratch ring served by IPv4/DiffServ pipeline. Along with a buffer handle, it copies the following metadata variables to scratch ring:

- `offset`—Offset of the start of packet data in the DRAM buffer.
- `input_port`—Identifier of an input port, where a packet was received.
- `packet_size`—Total size of a packet in bytes.

The scratch ring also conveys other metadata variables, not relevant to DiffServ blocks.

The IPv4/DiffServ functional pipeline begins with a dispatch loop source microblock (not shown in the figure). The block loads dispatch loop variables with metadata values received in a scratch ring. It also fetches the first 20 bytes of a packet header at offset in a DRAM buffer, and caches them inside a microengine. All subsequent blocks constituting the functional pipeline operate on dispatch loop variables and the cached packet header.

- The PPP decapsulation/classifier block removes L2-PPP header by updating `packet_size` and offset metadata fields. It also sets `header_type` based on the PPP fields. DiffServ Ingress pipeline do not support IPv6 packets, so the PPP decapsulation/classifier block drops them.
- The 6-tuple classifier takes selected fields of an IPv4 header and the input port as a lookup key. Ideally, the classification stage should be preceded by IP header sanity checks. However, it is not clear if this part of the code can be easily separated from the IPv4 forwarder.

  The lookup result contains `flow_id` (identifier of a packet flow) and `class_id` (a relative identifier of a target QoS queue). The class_id determines both the internal QoS class on the

switching fabric, as well as a target queue on the output port. If the switching fabric does not support QoS differentiation, the ingress IXP2400 or IXP2800 Network Processor simply ignores `class_id`. To accommodate color-aware SRTCM, the classifier may also set `color_id`.

If a hash table lookup fails, the classifier sets default values the above metadata variables. Depending on configuration, such packets can be then redirected to XScale core component (flow-cache model) or processed in microengine according to default `flow_id` and `class_id`. The classifier supports four outputs: one for metered packets, one for marked packets, one for packets handled in ME without marking/metering, and one for invalid packets (exception thrown to XScale).

- The SRTCM block takes `flow_id` as an index to a meter instance table. The microblock measures temporal flow characteristics (using `packet_size`), and divides packets into three conformance levels. It writes a conformance level to `color_id` variable, and updates relevant statistics in SRAM. A conformance level for a given `flow_id` can be configured with a drop or pass action. If a packet is allowed to pass, the microblock updates `flow_id` variable with the packet mark value—for example, DSCP value.
- A DSCP marker block writes `flow_id` into the TOS field and updates IP header checksum. A standalone marker allows bypassing SRTCM, if metering is not needed.
- The IPv4 forwarder validates an IP header. For valid packets, it performs Longest Prefix Match lookup on IP destination address. It stores the following information in dispatch loop variables:
  - identifier of an egress blade (`fabric_port`)
  - identifier of an output port on that blade (`output_port`)
  - identifier of next-hop data that can be used by L2 encapsulation on an egress blade (`next_hop_id`).

  On an IXDP2400 Advanced Development Platform, there is no ingress WRED. However, this block can be used in multi-blade configurations. If included, this block would calculate a queue number from `fabric_port` and `class_id`. In addition, the packet color_id selects the WRED instance. In case of congestion, WRED randomly drops packets and updates statistics. It does not change dispatch loop variables (except for `dl_next_block`).

The IPv4/DiffServ functional pipeline ends with a dispatch loop sink microblock (not shown in the figure). This block writes dispatch loop variables to SRAM metadata.

The sink block of a functional pipe stage sends enqueue messages to Queue Manager. The enqueue message contains an absolute `queue_id` constructed from `fabric_port` and `class_id`. The composite `queue_id` is also exchanged between the Queue Manager and the CSIX scheduler, inside enqueue/dequeue transition messages—refer to the Chapter 2, “System Data Structures and Design Choices” for details.

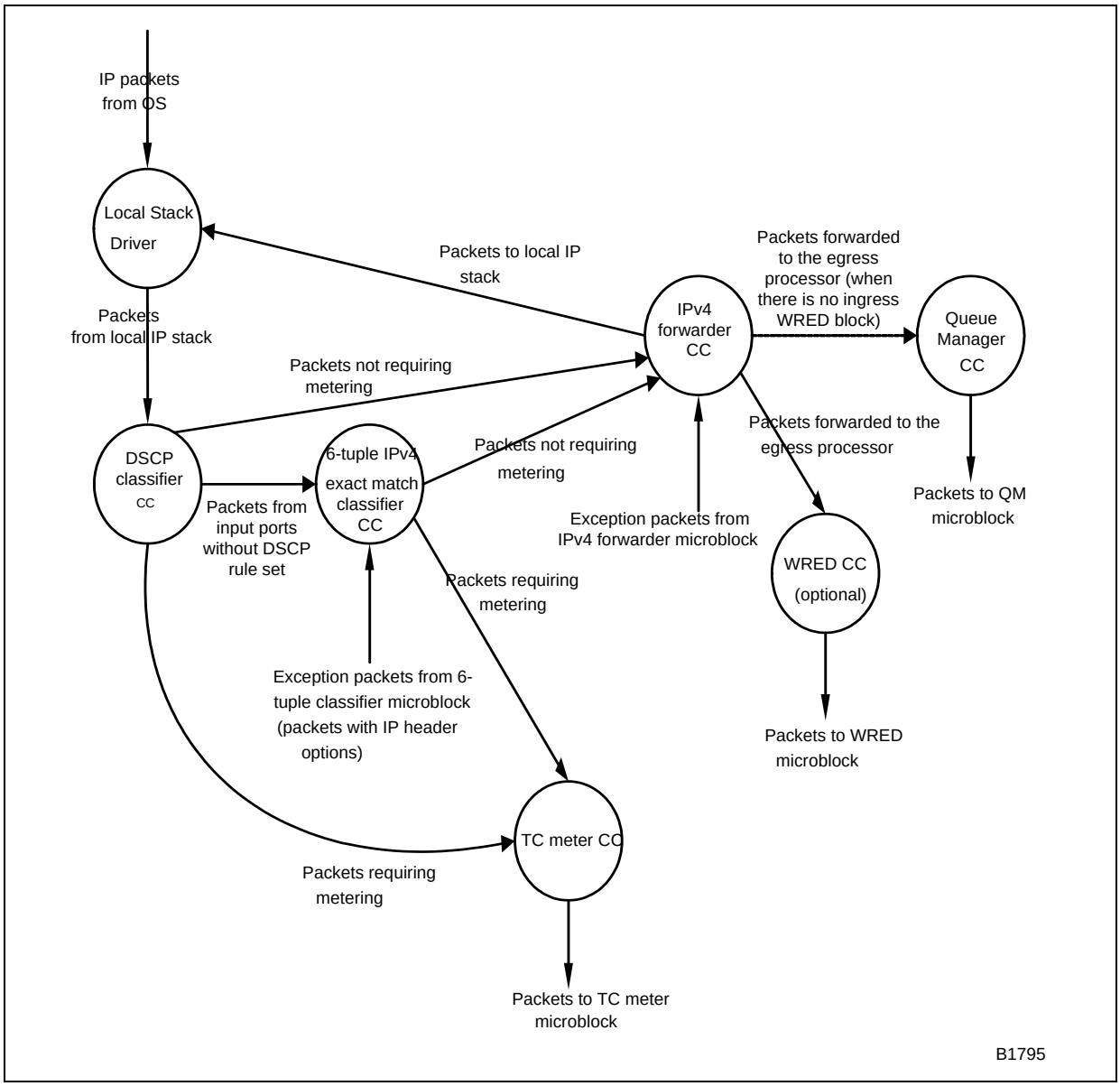
Once a packet is scheduled for transmission, it goes to the CSIX TX. The following metadata variables are transmitted over the switching fabric along with a packet:

- packet_size (in CSIX-L1 base header)—needed by egress CSIX RX
- class_id (in CSIX-L1 extension header)—needed by egress WRED
- flow_id (in TM header)—not needed, unless a DSCP re-classifier is used
- color_id (in TM header)—needed by egress WRED
- output_port (in TM header)—needed by egress WRED & POS TX
- next_hop_id (in TM header)—needed by POS encapsulation

8.5.1.1 Ingress Core Components

Figure 8-4 illustrates slow path in the POS DiffServ application. Packets coming from local IP stack are directed to the DSCP classifier core components. The core component does not receive any exception packets. It performs classification in the same way as its microblock. If a packet corresponds to an input port that does not have its classification rules, the packet is directed to the default output that is bound to 6-tuple exact match classifier core components. If a packet requires traffic conditioning, the core component passes it to TC meter core components. If the packet requires DSCP value remarking, the core component sets the new DSCP value in the IP header by itself. Next all the packets that do not need metering are passed to the IPv4 forwarder core components.

Figure 8-4. Slow Path for POS DiffServ Application



The 6-tuple exact match classifier core component receives local packets not classified by the DSCP core components. It also gets packets with IP header options sent from 6-tuple classifier microblock as exception packets. The 6-tuple core component classifies packets in the same way as its microblock. If the packet requires traffic conditioning, they are passed to the TC meter core components. If the packets only require DSCP remarking, the classifier core component sets the new DSCP value in the packet header. Next the classfier sends all the packets that do not need metering to the IPv4 forwarder.

The TC meter core components does not meter the packets coming from the classifier blocks, but it sends them to its microblock. This avoids implementing critical section shared between the core component and the microblock.

The IPv4 forwarder core components receives exception packets from its microblock. It also gets packets from the classifier blocks. It performs the same action on the received packets as the IPv4 forwarder microblock. It is it does LPM lookup and sets the lookup results in the packet meta-data. It passes the packet directed to the local IP stack to the Local Stack Driver. The packets forwarded to the egress processor are either sent to WRED core components (if ingress WRED is used) or to the QM core components. The forwarder core component also generates ICMP packets that are directed to the egress processor.

Both WRED core components and QM core components do not process the packets by themselves, but they pass the packets to their counterparts in microengines.

All the core components used on the ingress processor are responsible for initializing their microblocks and setting configuration tables used by the blocks. The core components expose API functions used by the System Application to configure the blocks.

Additional core components not shown in the figure are POS RX core components, CSIX TX core components and Scheduler core components. POS RX core component initializes Packet RX microblock and receives PPP LCP/IPCP control packet sent from the microblock as exception packets. The core component drops the packets. However in other applications the control packets can be captured by other component. The component also exposes functions for reading ingress POS interface state and statistics of the received packets.

CSIX TX core components initializes the CSIX TX microblock and implements a function for reading statistics of packets transmited over the switch fabric.

Scheduler core components initializes its corresponding microblock and fills up the configuration array used by the micrblock for WRR algorithm.

8.5.2 Egress Data Flow

8.5.2.1 Microblock Egress Pipeline

As per DiffServ MIB [RFC3270], a PHB program is configured for each output port and QoS class. The variables output_port and class_id are available in CSIX RX. Thus, the egress processor does not need to re-classify packets. However, the EF PHB program can start with SRTCM, while AF PHB typically begins with WRED.

The PHB entry point is derived from the class_id value. If class_id indicates a high-priority queue, a packet is passed on to TCM. Otherwise, a packet goes to WRED. Both microblocks send packets to PPP encapsulation and then to Queue Manager. This corresponds to a typical PHB

configuration, where TCM and WRED blocks are mutually exclusive. Basically, there is no point in doing WRED on a priority queue, since the average queue size should be close to zero. Conversely, a queue protected with WRED needs no rate-limiter.

Table 8-10 shows packet metadata transmitted along with a packet.

Table 8-10. MPLS, IPv4 and Diffserv Egress Dispatch Loop Variables

Metadata	CSIX RX	TCM	WRED	PPP encap	QM + Sched	POS TX
packet_ size	>	<	<			
flow_id	>	< >				
class_id	>		<		<	
color_id	>	< >	<			
output_ port	>		<		<	<
next_hop_id	>			<		

The egress datapaths starts with the CSIX receiver, which resembles CSIX frames and stores packets in DRAM buffers. It also restores metadata variables conveyed over the fabric. The block puts a buffer handle into a scratch ring served by IPv4/DiffServ pipeline. Along with a buffer handle, the microblock copies to scratch ring selected metadata variables, as specified in Table 8-5.

The egress functional pipeline begins with a dispatch loop source microblock (not shown in the figure). It reads scratch ring messages and restores dispatch loop variables. It also constructs flow_id variable for SRTCM block. Note that the ingress NP sets flow_id to a DSCP value. This is not enough to discriminate between EF PHBs configured for different output ports. For that reason, the DL source block overrides flow_id with a combination of output_port and class_id.

The TCM block uses flow_id, prepared by a DL source, to police EF traffic stream on a given interface. Only two conformance levels are used: in- and out-of-profile packets. The block is configured so that it drops excessive packets, while in-profile packets are placed in output queues.

The egress WRED block calculates a queue number from output_port and class_id. The remaining operations are same as for the ingress WRED block.

The PPP encapsulation block checks if next_hop_id is different than -1. If true, it adds the PPP header to the packet based on the header_type.

The IPv4/DiffServ functional pipeline ends with a dispatch loop sink microblock (not shown in the figure). This block does not need to update SRAM metadata descriptor, as the variables modified by TCM will be no longer used.

The data flow between egress Scheduler, egress QM and POS TX is the same as in [IXA DM].

8.5.2.2 Egress Core Components

The egress core components are:

- CSIX RX core components
- WRED core components
- TC meter core components
- QM core components

- Scheduler core components
- ATM/POS TX core components

The core components residing on the egress processor does not implemente a slow path. They are only responsible for initializing the egress microblocks and for setting configuration data strcutures used by the microblocks. In addition, the ATM/POS TX core components is responsible for initialisation and configuration of the ATM/POS framer device.

8.6 Sending Packets from Core Components to Microblocks

Some DiffServ core components do not process local and exception packets in slow path, but direct the packets to their microblocks (for example, TCM core components forwards packets to TCM microblock in order to avoid implementation of cirtical section between the core component and the microblock). In particular, a core component must be able to send packets to a microblock residing in the middle of a functional pipeline. For this reason the core component should send not only the SOP buffer handle but also an identifier of the microblock to which the packet is directed.b

Core components cannot write packets to the scratch ring preceding the microblock (it is the scratch ring used by the preceding microengine), because Xscale core cannot write atomically to a scratch ring more than a single LW and messages put to the scratch rings usually are more than 1 LW long. Therefore Xscale must use a separate scratch ring for sending packets to each microengine. However, if the functional pipeline runs on more than one microengine and it does not matter on which microengine the packet will be processed, only one scrach ring is needed per a functional pipeline.

If the DiffServ core components are run in different threads, writing to the scratch ring must be synchronized so as not to mismatch messages from different core components.

On the core component level, the packets are sent to a communication ID common for the whole communication from core components to micrblocks in IPv4/DIffServ pipeline. System Application is responsible for binding the communication ID to appropriate scratch ring and to register a packet handler function that builds messages to MEs and writes them to the ring.

Table 8-11 describes the format of messages exchanged between DiffServ core components and DiffServ functional pipeline..

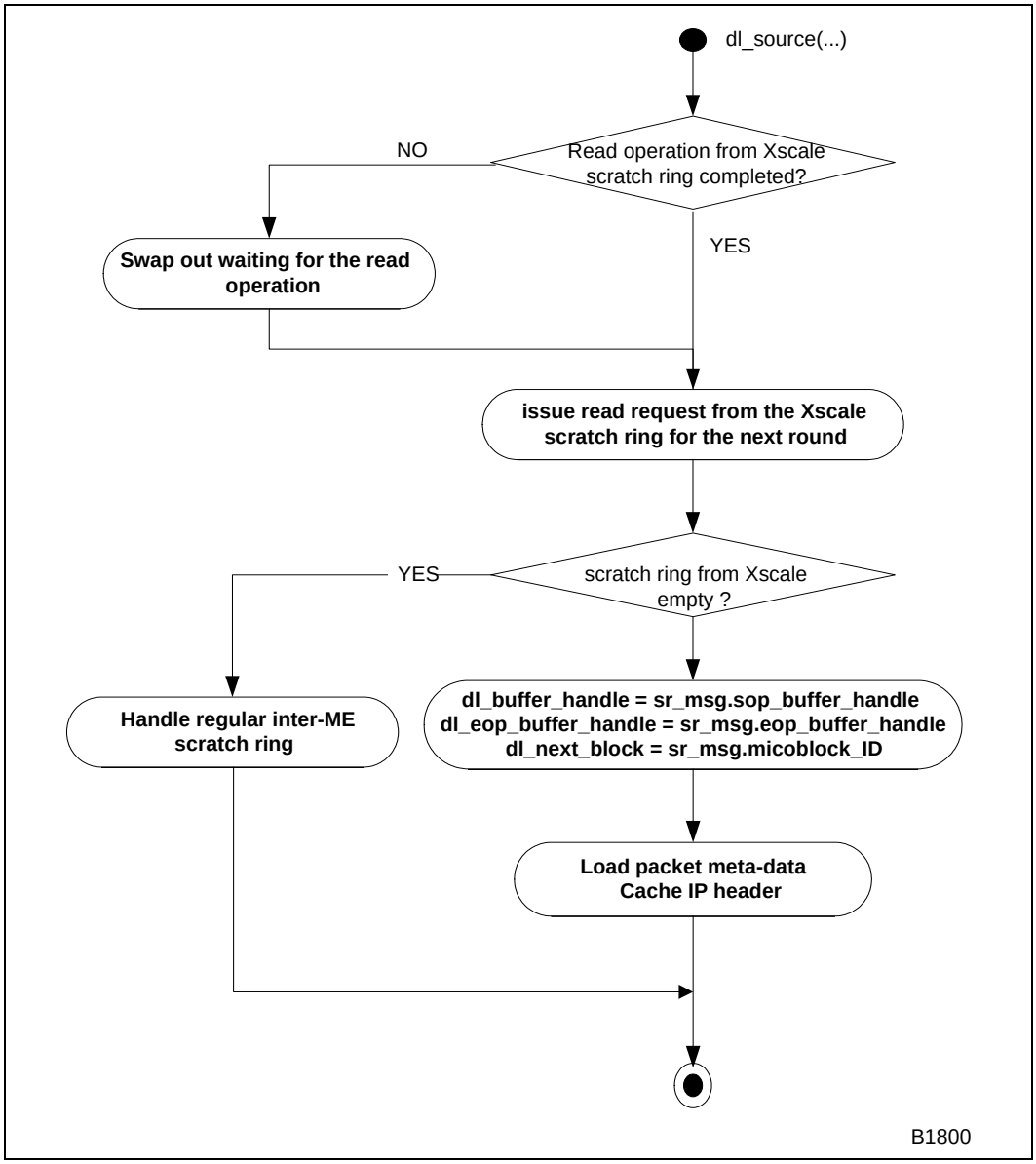
Table 8-11. DiffServ Core Components to Diffserv Pipeline Message Fields

LW	Field Name	Bits	Size (in bits)	Description
0	SOP buffer handle	31..0	32	Handle to the buffer containing the first part of the packet
1	EOP buffer descriptor	31..0	32	Handle to the buffer containing the last part of the packet. If the packet fits single buffer, the field contains zero value.
2	Reserved	31..8	24	Unused
	Microblock ID	7..0	8	Identifier of the microblock that should start processing the packet. The source macro uses tha value to set dl_next_block variable.

Assuming that the core components pass packets to microblocks rarely, only one thread on one ME running the functional pipeline reads the scratch ring from Xscale. This minimizes the number of empty read operation from the scratch ring.

Figure 8-5 illustrates the algorithm performed by the source macro called by this thread. First the macro checks whether the read operation issued in the previous dispatch loop round has completed. If not it swaps out waiting for the end of the operation. Next the macro checks if the ring is empty. If the ring is empty, it handles the scratch ring from the previous ME. Otherwise, it sets `dl_buffere_handle, dl_eop_buffer_handle and dl_next_block` variables according to the values in the message retrieved from the Xscale scratch ring. It also caches the packet meta-data and the packet header.

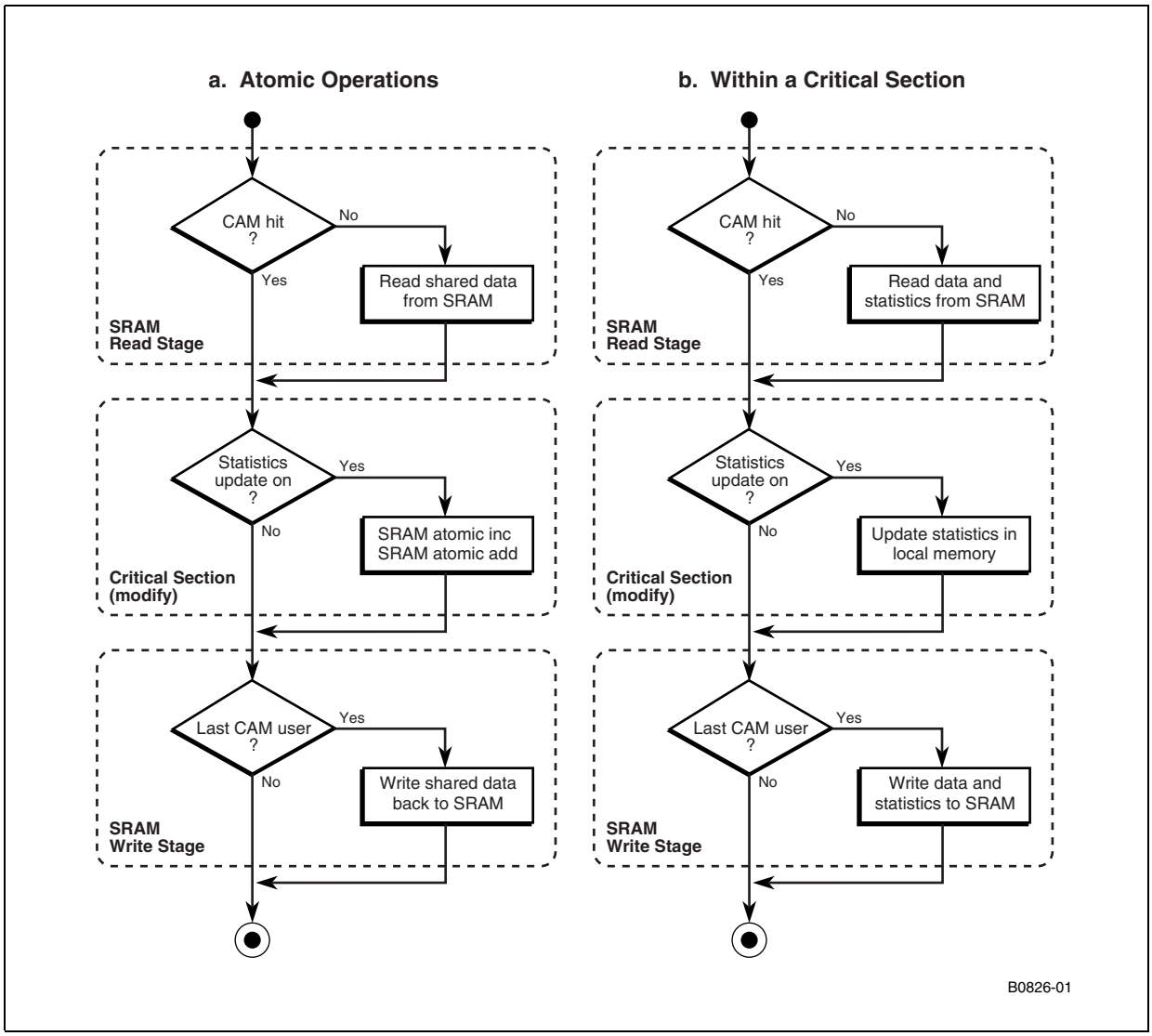
Figure 8-5. Handling Xscale Scratch Ring by the Source Macro



8.7 Statistics Handling

An IXP2400 SRAM controller supports atomic operations such as add or increment. It is quite tempting to use this feature for statistics updates. However, another option is possible if a microblock uses a folding technique (like SRTCM or WRED). Essentially, a microblock can include statistic updates within a critical section. Figure 8-6 illustrates both approaches.

Figure 8-6. Statistics Update—Atomic Operations and Within a Critical Section



Four memory references are the main drawback of the left-side approach (a). In addition, atomic operations have to be always performed, independently of CAM hit. On the contrary, the right-side algorithm (b) causes longer memory bursts. Moreover, this method always reads statistics counters, even if an update is not needed (turned off dynamically).

Nevertheless, it seems more beneficial to perform two long memory accesses than four short ones. This section provides a performance comparison of both approaches. Simulator-based experiments exhibit the following performance of SRAM controller:

- 1 atomic operation blocks SRAM controller for about 30 cycles.
- 1 long word memory access blocks SRAM controller for about 3 cycles. Every next long word in a memory burst blocks SRAM controller for another 3 cycles, that is:
  - 8 long words memory access blocks SRAM controller for about 24 cycles.
  - 16 long words memory access blocks SRAM controller for about 48 cycles.

From the above calculation, it follows that an atomic SRAM operation is as expensive as reading/writing a burst of 10 long words. Considering SRAM controller utilization, the approach (b) shall perform better if statistics data comprises less than 10 long words. In fact, both SRTCM and WRED blocks need 6 long words for statistics. The approach (b) shown in Table 8-12 can be even more advantageous, if CAM hits occur.

Table 8-12 shows a detailed comparison of both approaches for different usage scenarios, assuming 6 long words for counters. The cycle count overhead corresponds to one processed packet.

Table 8-12. Statistics Overhead at SRAM Controller

Scenario	Statistics Update		No Update Needed	
	All CAM Misses	Max CAM Hits	All CAM Misses	Max CAM hits
(a) using atomic operations[1]	63	60 3/8	3	3/8
(b) within a critical section	36	4 4/8	18	2 2/8
Gain: (b) – (a)	+ 30	+ 55 7/8	- 15	- 1 7/8

1. 3 cycles are always used to read the statistics counter location (one long word).

The method (b) behaves significantly better if statistics are updated. Due to non-linear queuing characteristics, the advantage can be greater than gain expressed in cycle counts. If a method (a) runs concurrently on 4 microengines in a functional pipe stage, the 60 cycle overhead per packet is enough to overload the SRAM controller command bus. As a result of command bus overflow, microengines are halted and performance drops drastically.

Overflows do not happen in method (b). This feature compensates deficiency caused by surplus memory reads if statistics update is not needed. For the above reasons, the method (b) is recommended and used in blocks implementing a critical section.

DiffServ for ATM Application 9

This section describes a DiffServ application for ATM implemented on two half duplex Intel® IXP2400 Network Processors connected to a CSIX switch fabric. It provides a high-level design overview and lists the different software components used to build this application.

The application described in this chapter is supported on the Intel® IXDP2400 Advanced Development Platform.

9.1 Hardware Architecture

The DiffServ for ATM application runs on the same hardware platform as plain ATM Application *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*. The platform consists of two IXP2400 processors. The ingress processor receives ATM cells from Media interface (ATM Framer), performs ingress DIffServ/IPv4 processing and sends the IP datagrams segmented into CSIX C-frames to the CSIX fabric.

The egress processor receives CSIX C-Frames from the fabric, reassembles these into IPv4 datagrams and performs egress DiffServ processing. Next the IPv4 datagrams are encapsulated into LLCSNAP packets, segmented into ATM cells and transmitted over the appropriate ATM physical port.

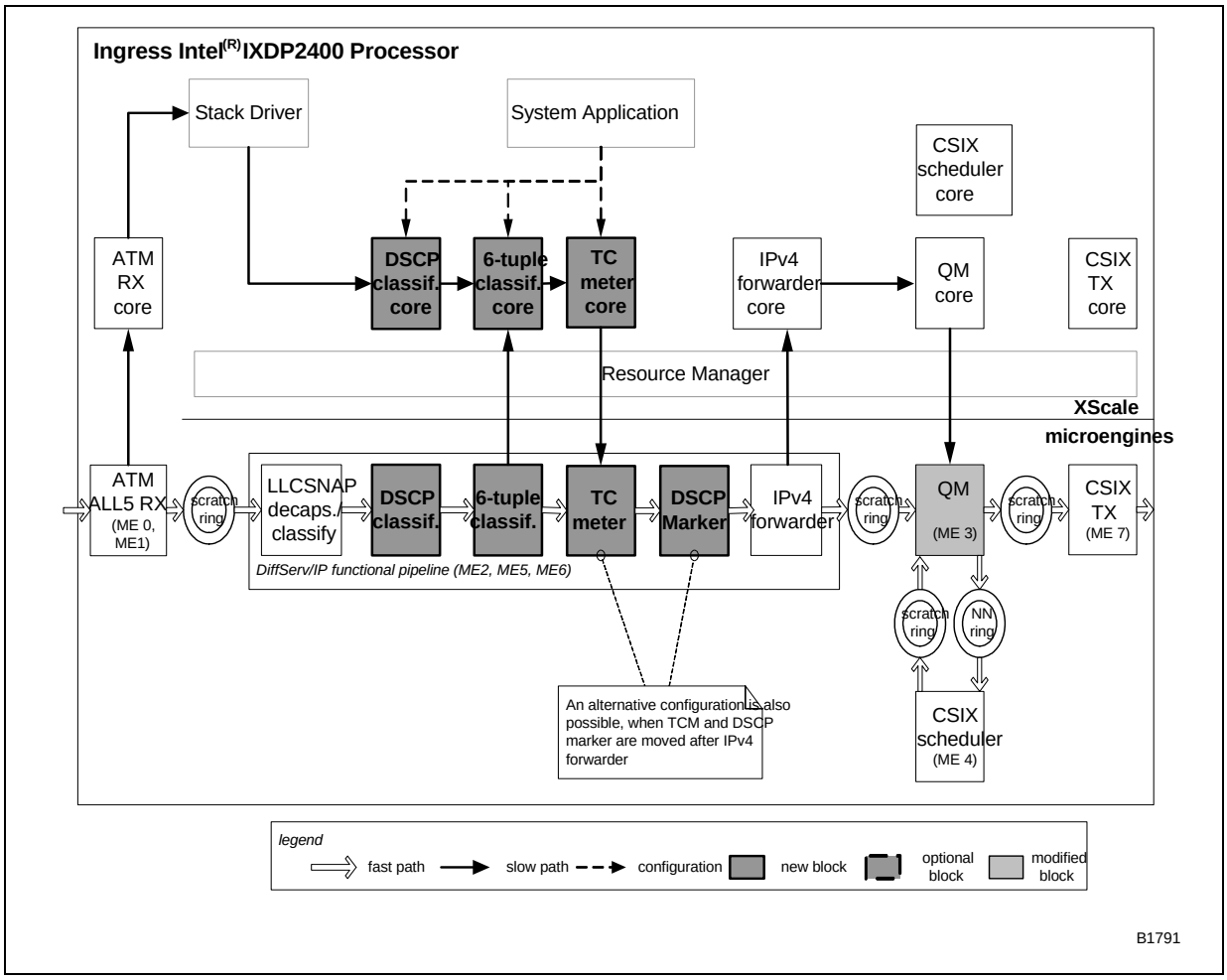
9.2 Software Architecture

9.2.1 Ingress IXP2400

Figure 9-1 illustrates the software architecture of ATM/DiffServ/IPv4 blocks on the ingress processor. The diagram shows mapping of functional blocks to microengines. The physical assignment determines inter-microengine communications, such as scratch rings or next neighbor registers. The arrangement affects also utilization of the internal Command bus and S-Push/Pull buses.

The DiffServ blocks (shaded boxes) extend an existing IPv4 reference design (white/clear boxes).

Figure 9-1. Software Architecture of ATM/DiffServ/IPv4 blocks on the Ingress Processor



9.2.1.1 Ingress Microblock Pipeline

9.2.1.1.1 ATM RX

The ATM RX block is the same microblock as used in the ATM application. It runs on two microengines. The block reassembles ATM cells coming from the media interface into AAL5 PDUs, writes them into DRAM and queues the packet buffer handle on a ME-ME scratch ring for processing by the next stage.

In DiffServ for ATM application, the DiffServ/IPv4 functional pipe executes in parallel on three microengines. The pipeline is organized in a dispatch loop, which starts with a 6-tuple classification and metering. In this way, packets dropped by a meter do not go through IP lookup. Alternatively, the TCM/DSCP blocks can be moved after the IPv4 forwarder. In such case, packets with invalid headers—for example, TTL expired, does not get unnecessarily metered. Both arrangements give the same performance in the worst case, when all packets have valid headers

and are always marked. The ingress dispatch loop can optionally contain WRED congestion avoidance (not shown in Figure 9-1). The ingress-side WRED accommodates multi-blade environments, and will not be implemented on Angel Island hardware platform.

9.2.1.1.2 LLCSNAP Decapsulation/Classify

The LLCSNAP decapsulation/classify microblock checks if the header type in the meta-data has been set to LLCSNAP. If the packet uses LLCSNAP encapsulation the block removes the LLCSANP header from the packet. It also classifies the packet into IPv4, IPv6, and so on. If the packet is not using the LLCSNAP encapsulation, the packet classification (`dl_next_block`) is done based on the value of the header type field in the packet meta-data. IPv4 packets are passed on to the 6-tuple classifier for further processing. IPv6 packets are dropped in this pipeline.

9.2.1.1.3 6-tuple Classifier

The rest of the function pipestage is nearly the same as for POS application. First packets go to DCSP classifier microblock that performs packet classification basing on input port and DSCP value carried in the packet header. If the packet matches a classification rule it is either directed to TC meter block or DSCP marker block. If the classification fails, the packet is directed to 6-tuple classifier microblock that performs an exact-match lookup on the IPv4 header and classifies the packets into QoS flows. In ATM scenario, each VC is associated with a single queue. IPv4 fowarder decides to which VC a packet is directed. For this reason, the class ID variable set in the 6-tuple classifier microblock is not used at egress.

After QoS classification the packets are metered by TCM implementing the metering algorithms described in [RFC2697] and [RFC2698]. Then DSCP marker updates TOS field in the IP header.

9.2.1.1.4 IPv4 Forwarder

The IPv4 Forwarder microblock validates the IP header as per RFC 1812. Invalid packets are dropped. Otherwise, a mircoblock performs Longest Prefix Match (LPM) on the IP destination address. The lookup result specifies destination where a packet should be forwarded.

9.2.1.1.5 Queue Manager

The ingress Queue Manager performs enqueue/dequeue operations on the hardware-assisted SRAM queues. The QM receives enqueue requests from the IPv4/DiffServ pipeline through a scratch ring. Another scratch ring is fed with dequeue requests from the CSIX scheduler. When the queue state changes between empty and non-empty, QM sends a transition message to the Scheduler (via Next Neighbor registers). After every dequeue operation, the QM passes a transmit request to the scratch ring served by TX microblock. All messages posted in scratch/NN rings have the same format as described in Section 2.6, “Interfaces Between the Various Microblocks” on page 34.

9.2.1.1.6 CSIX Scheduler

This CSIX scheduler selects constant-length packet segments (cframes) to be transmitted to the CSIX fabric. The scheduler employs Round Robin (RR) among the fabric ports and Weighted Round Robin (WRR) among the port queues. The scheduler handles also flow control messages received from the fabric. This block is not modified.

9.2.1.1.7 CSIX TX Microblock

The CSIX TX microblock receives transmit messages from the QM, and moves packet segments (cframes) into a transmit buffer. It also encapsulates cframe payload with a CSIX header, and a proprietary Traffic Manager (TM) header. The CSIX/TM headers convey metadata information to the egress processor. This block is not modified.

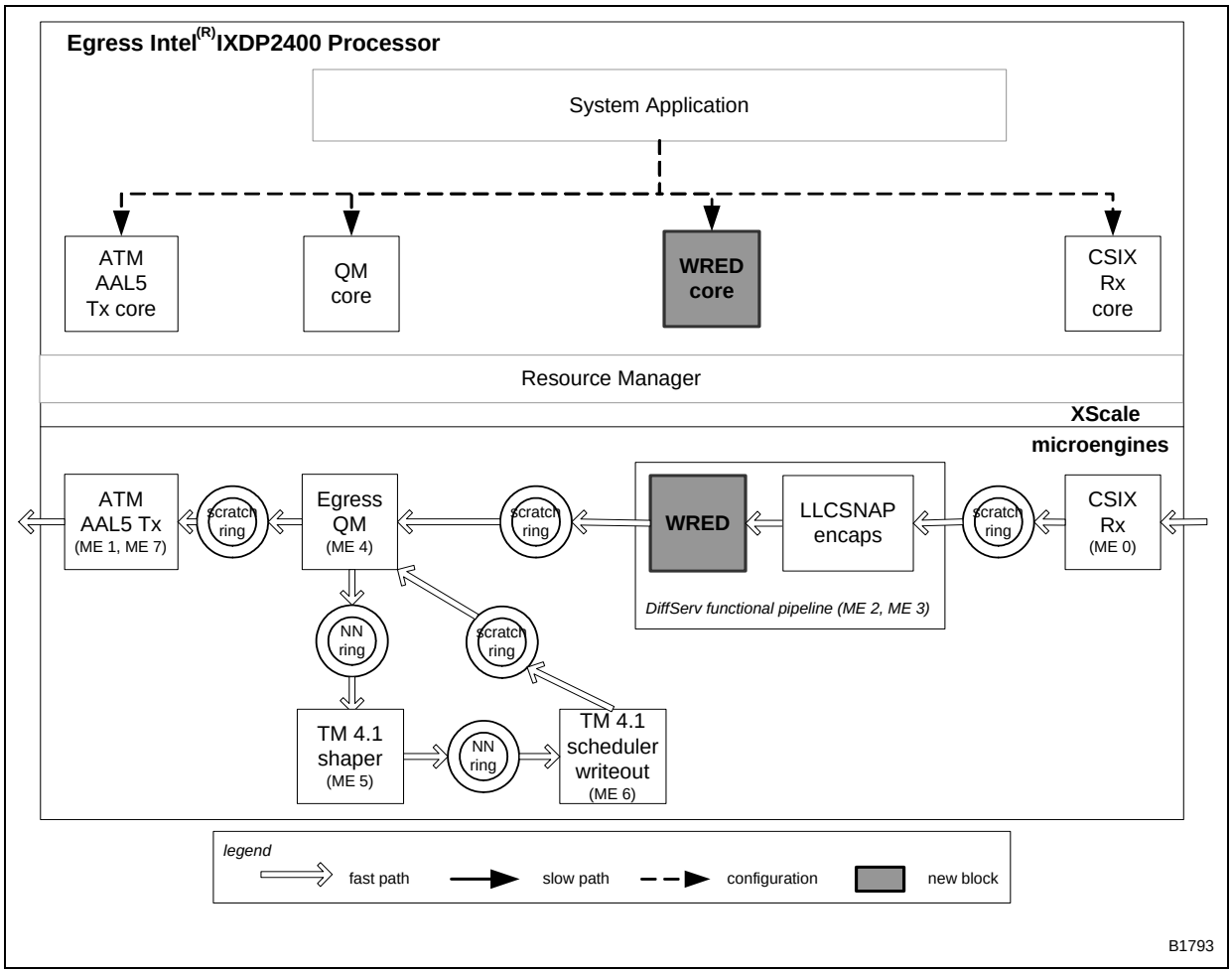
9.2.1.2 Ingress Core Components

DiffServ for ATM application uses the same set of ingress core components as desccribed for Diffserv POS application in Section 8.5.1.1, “Ingress Core Components” on page 116. The core components form the same slow path.

9.2.2 Egress IXP2400

Figure 9-2 illustrates the software architecture of DiffServ/IPv4/ATM blocks on the egress processor and mapping of functional blocks to microengines.

Figure 9-2. ATM, IPv4 and DiffServ—Egress Architecture



The CSIX RX block is the same as used in POS application. It reassembles cframe segments back into packets, and restores metadata information. Next, it passes a packet to the egress WRED microblock, using a scratch ring for communication.

The next two microengines run an egress-side DiffServ functional pipe stage. The pipe stage starts with the LLCSNAP encapsulation microblock. It is the same block as used in IPv4/ATM application. It adds the LLCSNAP header to the packet and sets VC queue number identifying the queue to which the Queue Manager shall put the packet. The encapsulation block uses the next hop id as an index into a table with layer-2 header information. This table contains both the LLCSNAP header as well as the Virtual Circuit (VC) Queue information for the packet.

The next element of the functional pipe stage is WRED congestion avoidance. WRED uses VC queue number as a queue identifier instead of combination of output port and class ID. At the end of the pipeline packets are passed to Queue Manager microengine.

The egress Cell Queue Manager block is similar to that used in plain ATM application. The only modification required by WRED is flushing queue idle timestamp on every transition of a queue state from non-empty to empty.

The rest of microblocks—Chapter 23, “TM4.1 Shaper and Scheduler Microblock” and Chapter 11, “ATM AAL5 TX Microblock” are used unmodified.

9.2.3 Performance Analysis

The analysis is same as for plain IPv4/ATM application (see Chapter 2, “OC-48 POS IPv4 Forwarding Application” and Section 4.2.4, “Performance Characterization” on page 58). In brief, IXP2400 operates at 600 MHz. For a min AAL5 packet of 2*53B, the packet inter-arrival time at OC-48 line rate is 210 microengine cycles. In order to maintain line rate for min packets, each stage of the pipeline cannot exceed this budget.

9.3 System Data Structures, Interfaces, and Resource Allocation

This section briefly depicts system-wide data structures used by DiffServ for ATM application. It also describes how system resources—for example, microengines, scratch rings, NN rings, memory regions, and so on, are allocated and used among the different microblocks. This chapter focuses on DiffServ blocks, while details on plain IPv4/ATM structures can be found in Chapter 4, “OC-48 ATM IPv4 Forwarding Application”.

9.3.1 Ingress System Resource Allocation

The allocation of ingress microengines is same as in the plain IPv4/ATM application - refer to Section 4.2.3, “Dispatch Loop” on page 56. Table 9-1 shows memory regions added by DiffServ microblocks. For performance reasons, all DiffServ structures are placed in SRAM. However, in a cost-oriented application it is recommended to put hash table in DRAM.

Table 9-1. Ingress IXP2400 Memory Usage

Item	Size per entry (in bytes)	Number of entries	Total SRAM used	Total DRAM used	Total scratch used
plain IPv4/ATM application	-	-	14.4 MB	64 MB	10 kB
6-tuple classifier hash table	32	64k	2 MB		
6-tuple classifier collision chains	32	32k	1 MB		
6-tuple classifier 64-bit stats	16	96k	1,5 MB		
TCM table	64	1k	64 kB		
TCM 64-bit stats.	32	1k	32 kB		
DSCP classifier table	8	16k	128 kB		
DSCP classifier 64-bit stats.	16	16k	256 kB		
Total			**19.3 MB**	**64 MB**	**10 kB**

Note: The hash table size can be much smaller. This is because the flow-cache model (with dynamic hash entries) does not scale to high-speed links. Thus, only statically configured hash entries are supported, and it is not likely that one configures all 64k of rules.

9.3.2 Egress System Resource Allocation

On egress IXP2400, one microengine is added to accommodate DiffServ PHBs, as compared with plain IPv4/POS application. Table 9-2 shows the modified microengine allocation.

Table 9-2. Egress IXP2400 Microengine Allocation

Microblock	ME#	Communication with previous block
CSIX RX	ME 0	Auto-push status from MSF
WRED + LLCSNAP encapsulation	ME 2, ME3	Scratch Ring
Egress Cell QM	ME 4	Scratch Ring
TM 4.1 Shaper	ME 5	Next neighbor
TM 4.1 Writeout / Scheduler	ME 6	Scratch Ring
AAL5 TX	ME1, ME7 (SPHY-4)	Scratch Ring

Table 9-3 shows memory regions added by DiffServ microblocks on egress processor.

Table 9-3. Egress IXP2400 Memory Usage

Item	Size per entry (in bytes)	Number of entries	Total SRAM used	Total DRAM used	Total scratch used
Plain IPv4/ATM application	-	-	2.13 MB	64MB	10kB
Queue Descriptors entry extension	16	1024	16 kB		

Table 9-3. Egress IXP2400 Memory Usage

Item	Size per entry (in bytes)	Number of entries	Total SRAM used	Total DRAM used	Total scratch used
WRED table	64	1024	64 kB		
WRED 64-bit stats.	32	1024	32 kB		
Total			2.24 MB	64 MB	10 kB

9.3.3 Buffer Handle

The DiffServ for ATM application uses the same buffer handle as described in Section 8.3.3, "Buffer Handle" on page 109 for DIffServ POS application.

9.3.4 Packet Metadata

The DiffServ for ATM application uses the same Packet Metadata layout as described in Section 8.3.4, "Packet Metadata" on page 109 for DIffServ POS application.

9.4 Interfaces Between the Various Microblocks

9.4.1 Inter-Microengine Messages

This section describes the interfaces between microengines on ingress and egress IXP processors for the ATM DiffServ application. The interfaces are described in terms of messages exchanged over scratch and NN rings. To ensure backward compatibility and easy migration, most of these interfaces are unchanged as compared with the IPv4 reference design described in Section 4.5, "Interfaces Between the Various Microblocks" on page 61. This section highlights only modifications.

9.4.1.1 AAL5 RX and Ingress DiffServ/IPv4 Functional Pipeline

The same as in plain ATM application—see Section 4.5.2, "Packet Processing Microengines and Cell Queue Manager" on page 61.

9.4.1.2 Ingress DiffServ/IPv4 Functional Pipeline and Ingress Queue Manager

See Section 4.5.2, "Packet Processing Microengines and Cell Queue Manager" on page 61.

9.4.1.3 Ingress Queue Manager and Ingress Scheduler

See Section 4.5.3, "Cell Queue Manager and CSIX Scheduler" on page 61.

9.4.1.4 Ingress Queue Manager and CSIX TX

See Section 2.6.4, "Cell Queue Manager and CSIX TX" on page 36.

9.4.1.5 CSIX RX and Egress DiffServ Pipeline

The interface between the CSIX RX pipe-stage and the egress DiffServ functional pipeline is the same as in plain ATM application. See Section 4.5.5, “CSIX RX and LLCSNAP Encapsulation” on page 62

9.4.1.6 Egress DiffServ pipeline and Egress Cell Queue Manager

Same as interface between ingress DiffServ/IPV4 functional pipeline and the Ingress Queue Manager. See Section 4.5.6, “LLCSNAP Encap and Cell Queue Manager” on page 62 for details.

9.4.1.7 Egress Cell Queue Manager and TM4.1 Shaper

See Section 4.5.7, “Cell Queue Manager and RR Scheduler for ATM” on page 62.

9.4.1.8 TM4.1 Shaper and TM 4.1 Writeout/Scheduler

See TM4.1 Shaper and Scheduler.

9.4.1.9 RR Scheduler and Egress Cell Queue Manager

Not changed—see Section 4.5.8, “RR Scheduler to Cell Queue Manager” on page 63.

9.4.1.10 Egress Queue Manager and AAL5 TX

Not changed—see Section 4.5.9, “Cell Queue Manager and AAL-5 TX” on page 63.

9.4.2 Ingress Dispatch Loop Variables

The DiffServ microblocks running in Ingress ATM DiffServ pipeline use the same dispatch loop variables as in Ingress POS DiffServ pipeline specified in Section 8.4.2, “Ingress Dispatch Loop Variables” on page 112.

9.4.3 Egress Dispatch Loop Variables

Unlike the DiffServ POS application, WRED should use VC queue number as a logical queue identifier. In order to change the WRED microblock, the egress DiffServ dispatch loop sets class ID variable to VC queue number and the output port variable to 0. The color ID used by WRED microblock is left unaltered—it is equal to the value set by the TCM block on ingress. The rest of egress microblocks uses the same dispatch loop variables as in plain ATM/IPv4 application.

MPLS Application 10

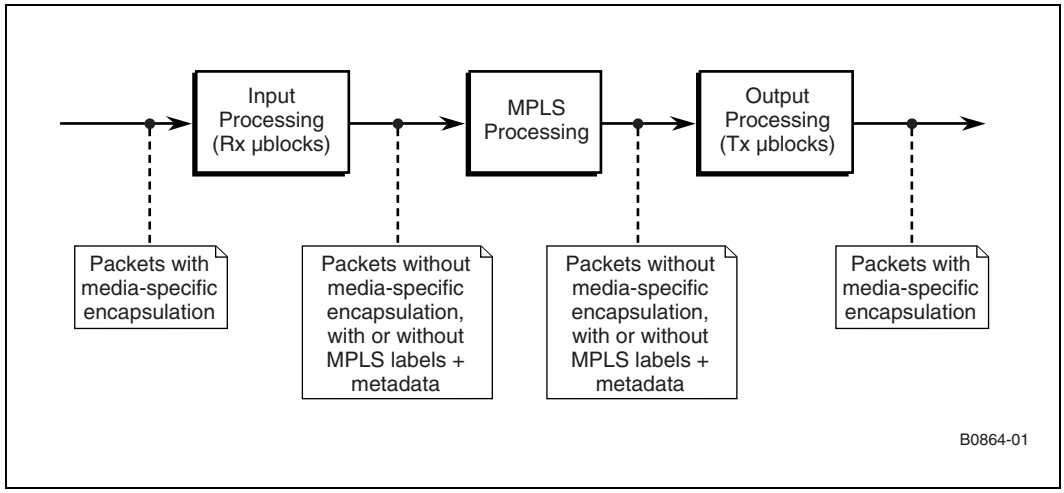
This chapter provides a high-level design overview of an MPLS forwarding application and lists the different software components used to build this application.

The application described in this chapter is supported on the Intel® IXDP2400 Advanced Development Platform.

10.1 Input/Output Media Independence

The MPLS forwarder building blocks design is based on the assumption that MPLS processing is separated from the packet receive and transmit microblocks. This separation hides the media-specific encapsulation details from the MPLS forwarder and enables MPLS forwarder building blocks reuse for Ethernet, POS and ATM. This concept is shown in Figure 10-1.

Figure 10-1. MPLS Flow Processing



The input processing stage receives packets with media-specific encapsulation, e.g. PPP for POS or 802.3 for Ethernet. It handles all the media-specific details, including any encapsulation processing and any re-assembly, for example in case of ATM. The output from this stage is a labeled or un-labeled packet without media encapsulation, and metadata that is needed for correct MPLS processing, e.g. incoming interface number. The implementation of this stage is out of scope of this documentation.

The MPLS processing stage begins with the classification of packets into FECs based on the attributes in the IP packet or incoming label and incoming interface. Next, it handles any MPLS specific processing of the received packet, e.g. creation, management and removal of the label stack, exception generation, TTL processing, etc. The output of this stage is an IP packet with or without label(s) and any metadata that is required by next stage to dispose off the packet correctly.

The details of MPLS processing depend on the role of the router in MPLS domain and are described further.

During output processing, packets received from the MPLS processing stage are put in appropriate media encapsulation and transmitted. The implementation of this stage is out of scope of this documentation.

10.2 MPLS Forwarder Decomposition

The MPLS Forwarder operation depends on the role of the router in the MPLS domain.

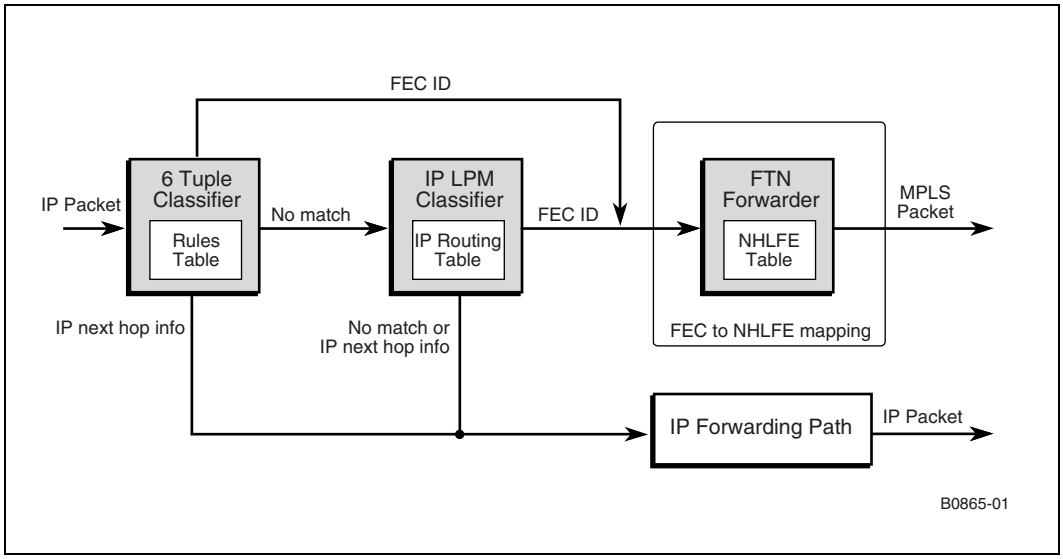
An MPLS router can work as the following types of forwarders:

Ingress LER	placed at the edge of an MPLS domain, receiving unlabeled IP packets, labeling them and sending to an MPLS next hop	Section 10.2.1, “Ingress LER Generic MPLS Forwarder” on page 132
LSR	placed in the middle of an MPLS domain, receiving labeled packets, swapping the labels and sending the packets to an MPLS next hop	Section 10.2.2, “LSR Generic MPLS Forwarder” on page 133
Egress LER	placed at the edge of an MPLS domain, receiving labeled packets, stripping off the labels and sending the packets to an IP next hop	Section 10.2.3, “Egress LER Generic MPLS Forwarder” on page 134

10.2.1 Ingress LER Generic MPLS Forwarder

The MPLS forwarder operation on an ingress LER is shown in Figure 10-2.

Figure 10-2. MPLS Forwarding: Data Path for Ingress LER



Incoming IP packets are validated as specified in [RFC1812]. Then they are classified according to forwarding equivalency classes (FECs). FECs can be defined by statically provisioned filters comprising multiple-field access control lists or IP 6-tuple, and IP destination address longest prefix match (LPM). The multi-field classification should be performed first. It the lookup fails, the longest prefix match is applied. Up to this point the packet processing is common for both IP and

MPLS. The 6-tupple Classifier Rules or IP Routing table entries contain flags marking them as belonging to either IP or MPLS forwarding path. Therefore a match against each entry determines whether further packet processing will be performed by IP or MPLS forwarders.

If a packet matches an MPLS FEC, the result of the rule or LPM lookup is an MPLS FEC identifier (FEC ID). Next, the packet with its meta data containing the FEC ID is passed to the FTN Forwarder block.

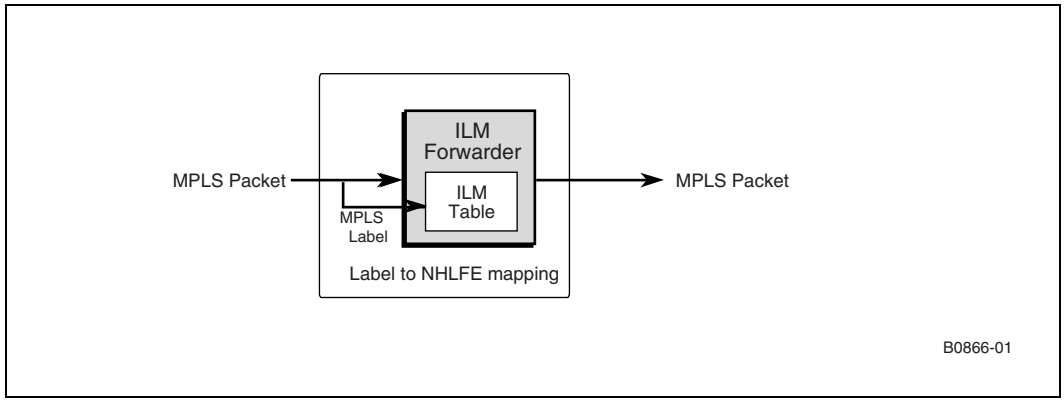
The FTN Forwarder uses the FEC ID from the packet meta-data as an index into the NHLFE table, to obtain a next-hop label forwarding entry assigned to this FEC. This can be either a regular entry, or an NHLFE set pointing to multiple NHLFE entries. In the latter case, only one entry is chosen by some additional load-balancing algorithm. A regular NHLFE contains information about the MPLS next hop, outgoing interface, and initial label(s). It is used by the FTN Forwarder, which appends the initial label to the beginning of the IP packet and puts the next hop information into the packet's meta data before passing it to a transmitting block.

According to the MPLS forwarder requirements, the 6-tuple and LPM classifiers shall be reused between the IP and MPLS components; therefore they are grayed and separated from the FTN Forwarder in Figure 10-2. In consequence, they have to be MPLS-aware, that is, the 6-tuple Rules and IP Routing table entries have to be distinguishable as either IP or MPLS entries.

10.2.2 LSR Generic MPLS Forwarder

At an MPLS transit node, incoming packets are classified by looking up the pair (top-most label, incoming interface) in the Incoming Label Map table. This operation is performed by the ILM Forwarder microblock. The ILM table entries contain the NHLFE information, the same as described for the ingress FTN Forwarder (outgoing interface, outgoing label, next-hop info). The ILM Forwarder performs a label swap or swap-push operation on the label stack and passes the MPLS packet to a transmitting microblock, together with meta-data specifying the next hop and outgoing interface. Figure 10-3 illustrates this functionality.

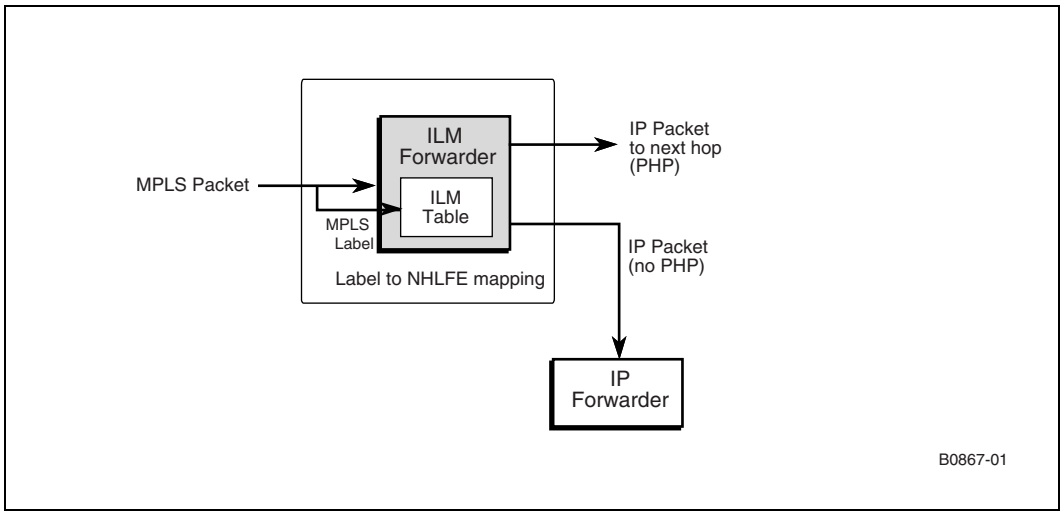
Figure 10-3. MPLS Forwarding: Data Path for LSR



10.2.3 Egress LER Generic MPLS Forwarder

The MPLS forwarder operation on an egress LER is shown in Figure 10-4.

Figure 10-4. MPLS Forwarding: Data Path for Egress LER



An egress node of an MPLS network receives labeled packets and performs on them topmost label lookups in the same way as in a transit LSR. However, an egress ILM entry indicates that all labels should be popped, and does not specify an outgoing label. Furthermore, the packet can be treated in two ways. In case of penultimate hop popping (PHP), the resulting unlabeled IP packet is passed directly to a transmitting microblock to be sent to its next hop over the outgoing interface specified in the ILM entry. Otherwise, the IP packet is passed to the local IP forwarder microblock for route lookup and forwarding.

10.2.4 MPLS Forwarder Building Blocks

The discussion from Sections 10.2.1, 10.2.2 and 10.2.3 implies that implementation of the MPLS forwarder functionality requires the following building blocks:

- FTN Forwarder - for ingress LERs
- ILM Forwarder- for LSRs and egress LERs

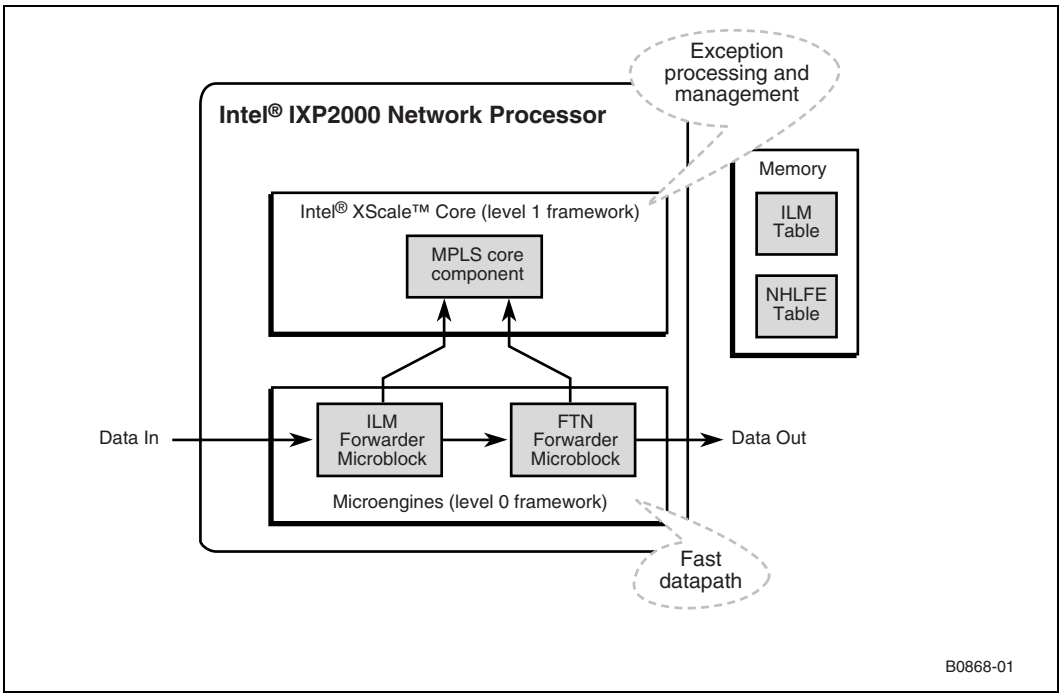
In practice, most MPLS nodes can receive unlabeled as well as labeled packets, and transmit labeled and unlabeled packets. Therefore, the MPLS forwarder should comprise components for all types of nodes - ingress and egress LERs, and transit LSRs.

The FTN Forwarder for ingress LERs depends on the IP 6-tuple classifier and IPv4 LPM microblocks. They are functional parts of other building blocks, and are described in their respective chapters of this document.

According to the IXA Portability Framework, the MPLS forwarder building block design follows the two-level software architecture. Figure 10-5 illustrates the two-level software architecture

.

Figure 10-5. MPLS Forwarder Building Block



The data plane microblocks (ILM Forwarder and FTN Forwarder) are initialized and managed by a common core component running on the XScale processor. The MPLS Core Component is responsible for adding and deleting entries in the ILM and NHLFE tables. It also processes exception notifications from the MPLS forwarder microblocks. Section 10.4.3, "MPLS Forwarder Core Component Overview" on page 143 describes the details of the MPLS Core Component design and operation.

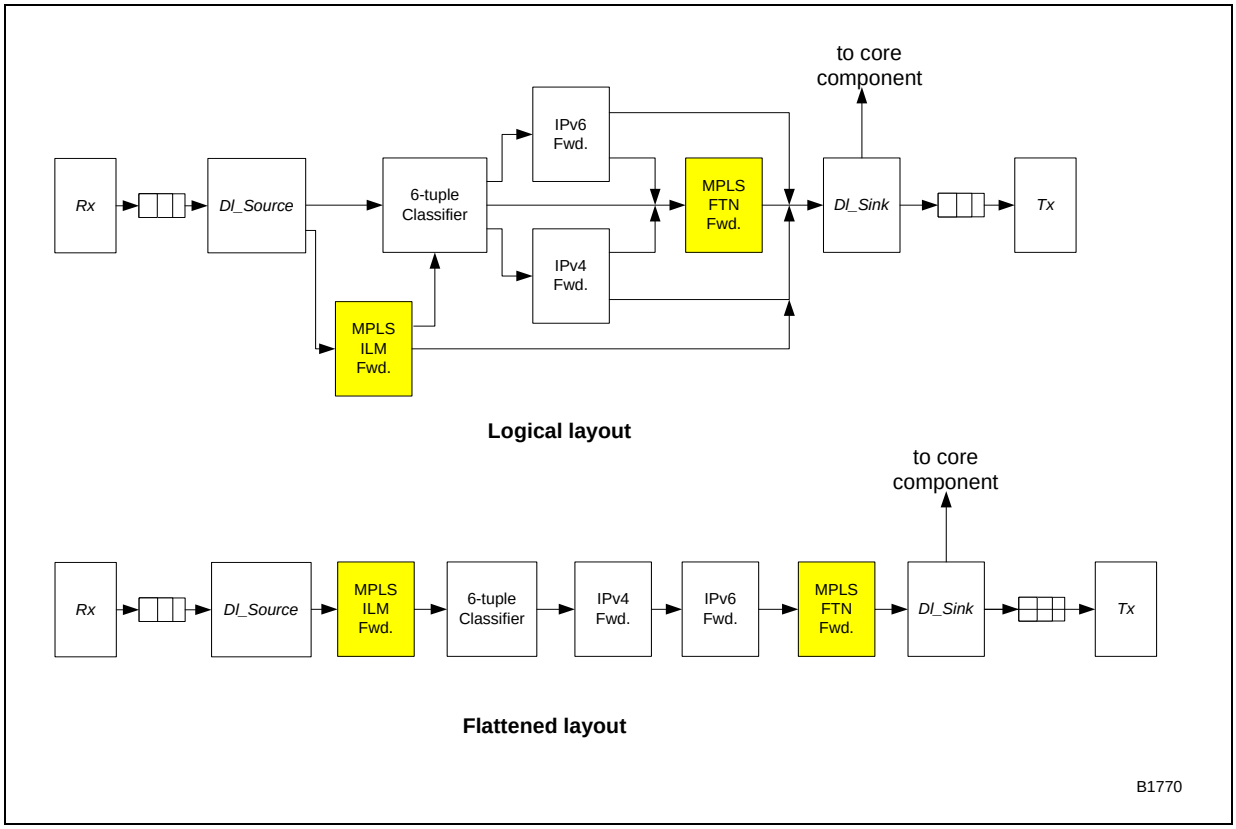
10.3 Cooperation with IP and QoS Microblocks

As it was mentioned in Section 10.2.1, "Ingress LER Generic MPLS Forwarder" on page 132, the FTN Forwarder reuses the LPM classification implemented in the IPv4 forwarder. Therefore, these two microblocks have to be combined on the same pipeline. Moreover, some MPLS nodes can be at the same time ingress LERs (for example, for locally connected hosts) as well as transit LSRs. This requires combining IP and MPLS forwarders on the same pipeline. This section shows the required ordering of IP and MPLS microblocks on the same pipeline (one thread).

10.3.1 IP and MPLS Functional Pipeline

Figure 10-6 illustrates the layout of IP and MPLS microblocks common for the ingress LER, LSR and egress LER.

Figure 10-6. Universal IP and MPLS Microblocks Layout



The functional pipeline starts with a Dl_Source microblock, which reads buffer handles from the receiving scratch ring, and populates the dispatch loop variables. One of these variables (`dl_header_type`), set by the RX microblock, specifies whether the received packet caries an IP or MPLS frame.

On an ingress LER, the Dl_Source block passes IP packets to the 6-tuple Classifier microblock, implementing the 6-tuple classification on behalf of the MPLS forwarder. In the case of a match, the packet will be passed to the FTN Forwarder. Otherwise, the next block will be either IPv4 or IPv6, depending on the packet. (The IPv4 and IPv6 mirocblock interactions are more complex than those shown in Figure 10-6).

Each of the IP microblocks has to implement an LPM function operating on an MPLS-aware routing table. The MPLS-awareness means that IP Routing table entries can be populated both by the IP routing protocols and MPLS control protocols. In the case of a match, the LPM lookup result specifies both the next block to execute and next hop id. If the next block variable points to the FTN Forwarder, the next hop id has a meaning specific to MPLS.

The pipeline ends with a Dl_Sink microblock. It passes the packet either to an appropriate core component in case of an exception set by one of the previous blocks, or to the transmitting scratch ring.

Either the IPv4 or IPv6 forwarder microblock can be removed from the above pipeline, if the node does not forward IPv4 or IPv6 traffic. However, at least one of them has to be placed before the FTN Forwarder block, to provide the LPM functionality.

On a transit LSR, the Dl_Source microblock recognizes MPLS packets by their L2 protocol number, present in the packet's meta-data, and sets the dispatch loop next block variable to point to the ILM Forwarder. The ILM Forwarder performs the MPLS label lookup in the ILM table to find an entry specifying label stack operation and the next hop for the packet. According to the lookup result, the ILM Forwarder can drop the packet, or pass it to the Dl_Sink microblock for sending to the TX microblock.

On an egress LER, the Dl_Source microblock recognizes MPLS packets by their L2 protocol number, present in the packet's meta-data, and sets the dispatch loop next block variable to point to the ILM Forwarder. The ILM Forwarder pops MPLS labels from the incoming MPLS packets, and forwards them further as native IP datagrams. However, two cases are possible.

In the case of penultimate hop popping, the packet is sent directly to the IP next hop specified in the entry pointed to by the result of the ILM table lookup. Therefore, after performing the pop operation, the ILM Classifier passes the packet to the Dl_Sink microblock for sending to the TX microblock.

In the case of ordinary popping, the ILM Forwarder performs the pop operation and passes the IP packet to the IPv4 forwarder for normal route determination.

Placing the ILM Forwarder before the IP forwarder eliminates necessity of looping back the packet to the beginning of the pipeline.

10.3.2 TTL Processing

[RFC3032] describes rules for TTL processing in MPLS networks. [RFC3443] updates these rules and ties together the tunnel terminology for MPLS support of Differentiated Services, introduced in [RFC3270], with TTL processing in hierarchical MPLS networks.

[RFC3270] defines three tunneling models:

- pipe—use outgoing per-hop behavior (PHB) to service the packet, leave exposed PHB information untouched. This model is used to hide intermediate MPLS nodes between LSP ingress and egress from the DiffServ perspective.
- short-pipe—use exposed PHB information to service the packet, leave exposed header untouched. This is a variation of the pipe model, in which the LSP egress outgoing interface uses the downstream cloud DiffServ policies. The short-pipe model is especially suitable when combined with penultimate hop popping.
- uniform—use the outgoing PHB to service packet, override exposed header with outgoing PHB. In this model all tunnel nodes are visible from the DiffServ perspective. The DiffServ information is always encoded in the outer-most label.

[RFC3270] requires that all routers implement the pipe model, while others are optional.

The Point Reyes MPLS forwarder supports all tunneling models described above.

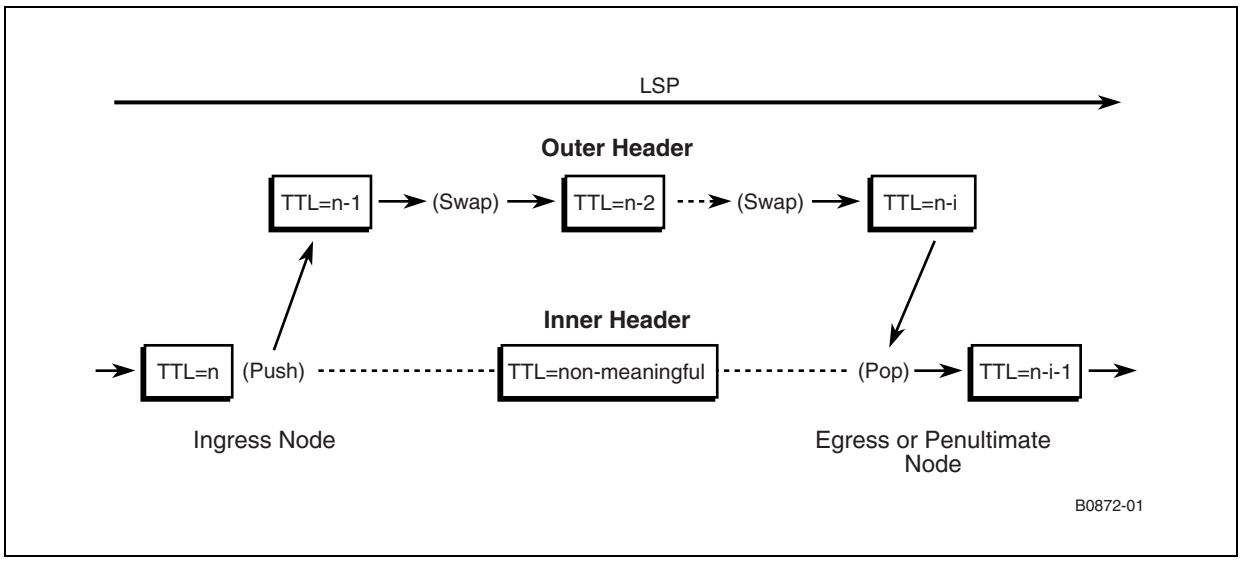
This section summarizes TTL processing rules for different tunneling models, consistent with [RFC3443]. The MPLS forwarder implementation conforms to those rules.

10.3.2.1 TTL Processing in Different Tunneling Models

[MPLS-TTL] presents TTL processing rules for different tunneling models, consistent with corresponding DiffServ information processing (consistent with [RFC3032]). The MPLS forwarder implementation conforms to those rules. They are summarized in this section.

Figure 10-7 illustrates the TTL processing for the Uniform model MPLS LSP (with or without PHP).

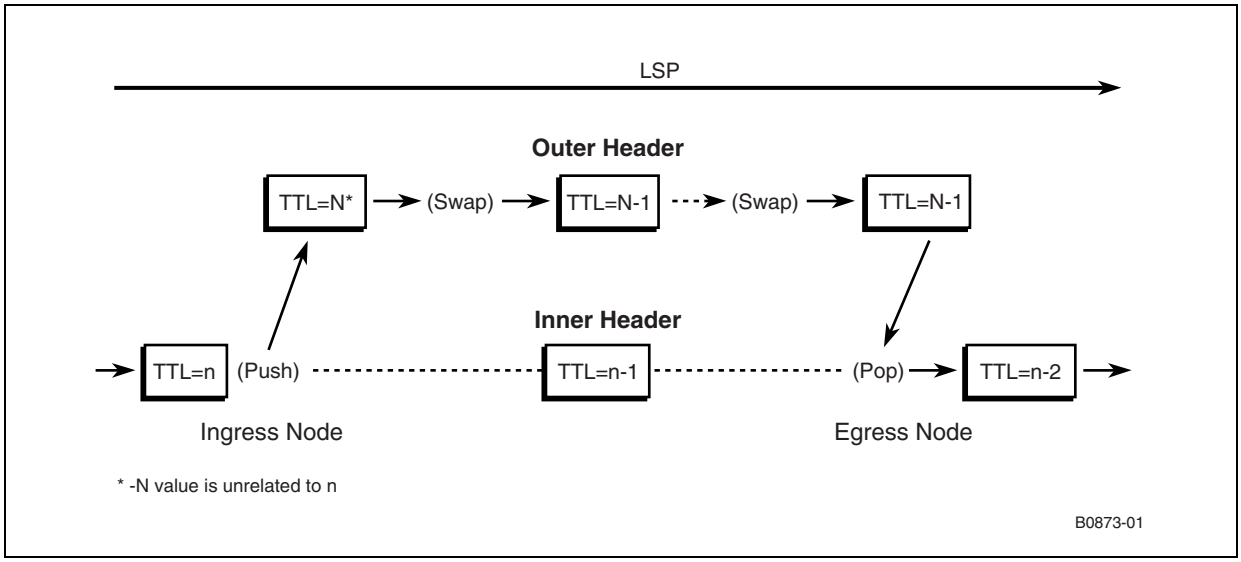
Figure 10-7. TTL Processing for Uniform Model LSPs



Note: The inner and outer TTLs of the packets are synchronized at tunnel ingress and egress.

Figure 10-8 illustrates the TTL processing for the Short Pipe model LSPs without PHP.

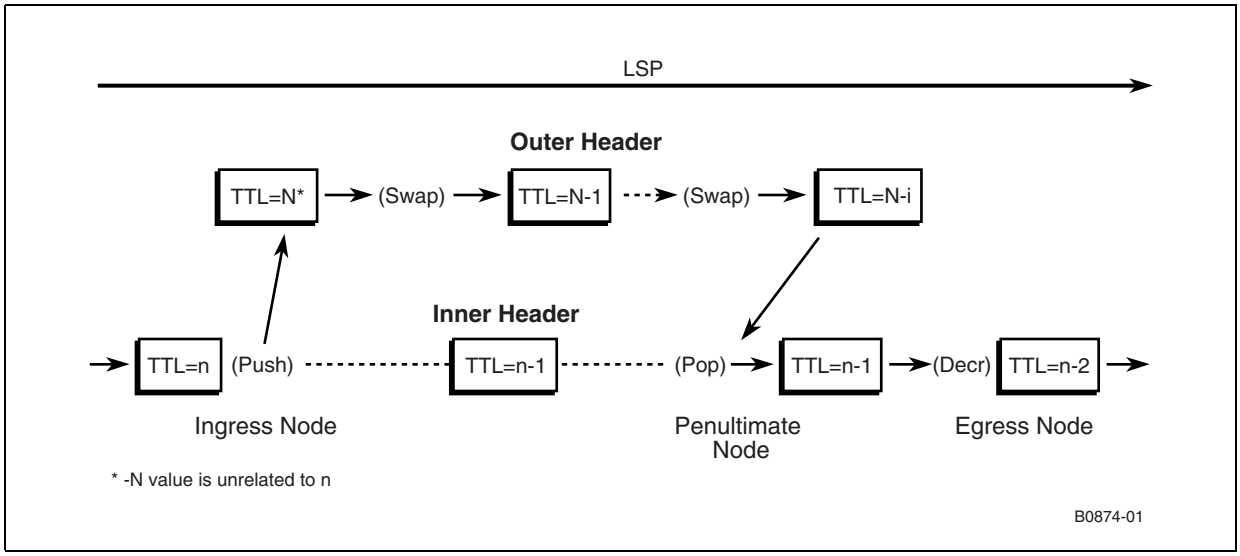
Figure 10-8. TTL Processing for Short Pipe Model LSPs without PHP



The Short Pipe model was introduced in [RFC3270]. In the Short Pipe model, the forwarding treatment at the egress LSR is based on the tunneled packet as opposed to the encapsulating packet.

Figure 10-9 shows TTL processing for the Short Pipe model with PHP.

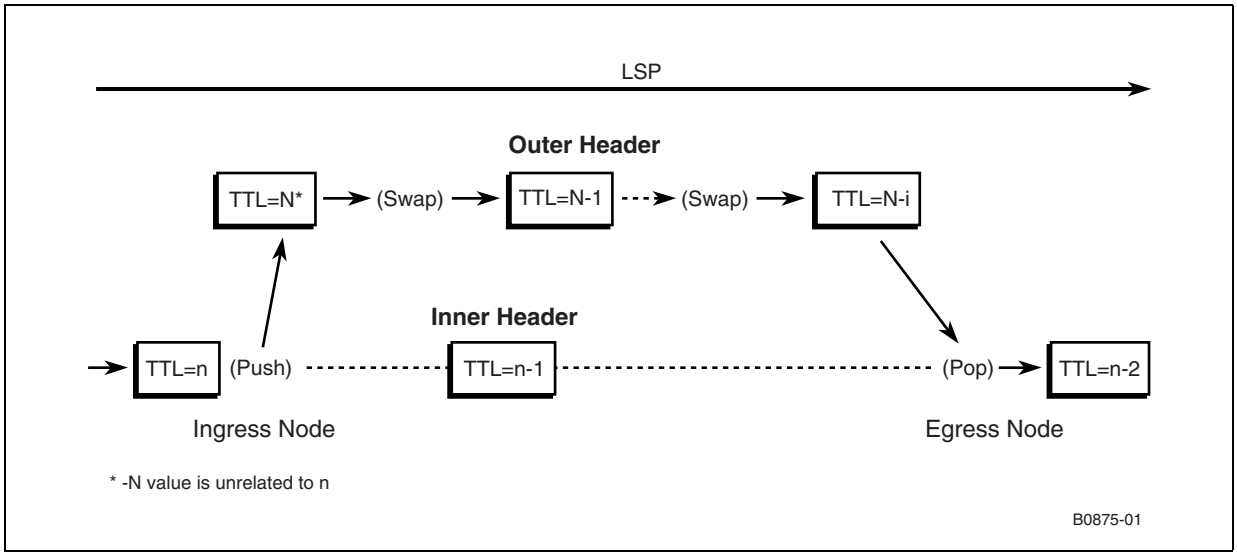
Figure 10-9. TTL Processing for Short Pipe Model LSPs with PHP



Since the label has already been popped by the LSP's penultimate node, the LSP egress node just decrements the header TTL. Also note that at the end of the Short Pipe model LSP, the TTL of the tunneled packet has been decremented by two either with or without PHP.

Figure 10-10 shows TTL Processing for the Pipe Model LSPs (without PHP only).

Figure 10-10. TTL Processing for Pipe Model LSPs without PHP



From the TTL perspective, the treatment for a Pipe model LSP is identical to the Short Pipe model without PHP.

10.4 Data Plane Architecture Dependencies

10.4.1 Target HW Architecture

The Intel® Internet Exchange Architecture Software Development Kit (IXA SDK) building blocks are targeted primarily at the dual IXP blade architecture shown in Figure 10-11.

In the above blade architecture, the ingress IXP processor on a blade receives data packets from an external interface (Ethernet, POS, ATM), processes them and sends them to the backplane (e.g. CSIX switching fabric). The egress IXP processor receives data packets from the backplane, processes them and sends them out through an external interface (Ethernet, POS, ATM). This scenario imposes division of the software building blocks into an ingress and egress data path.

Figure 10-11. Dual NP Blade Architecture

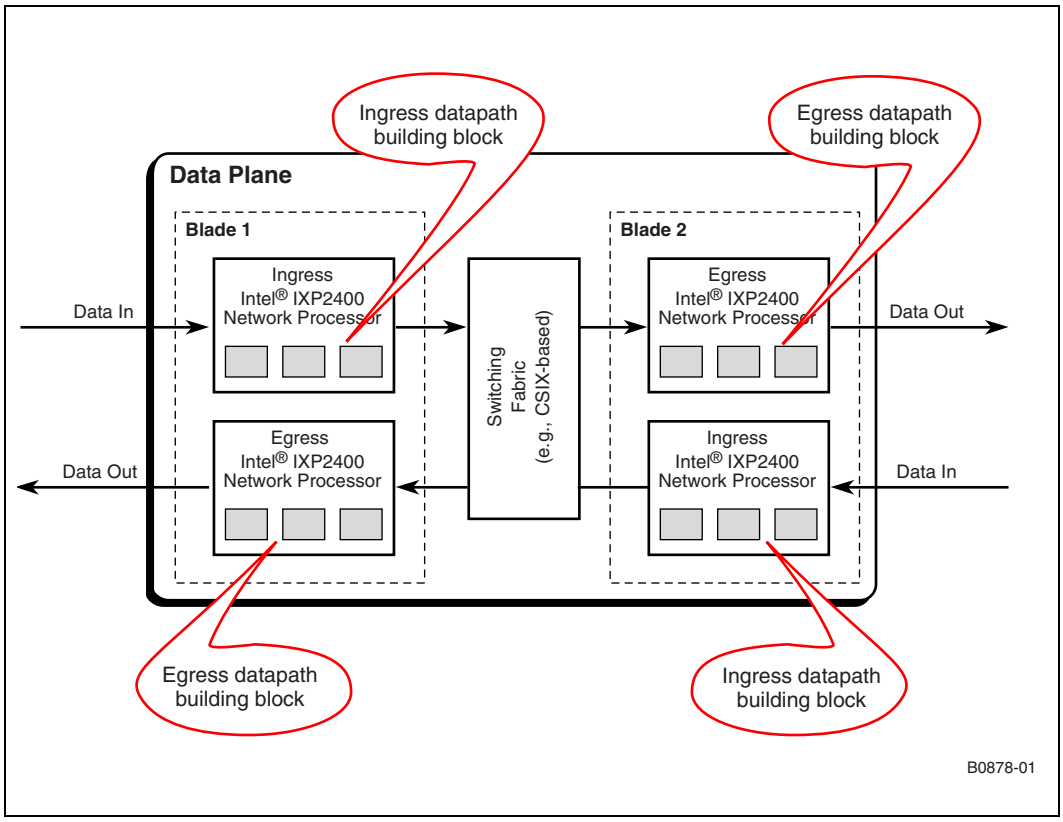
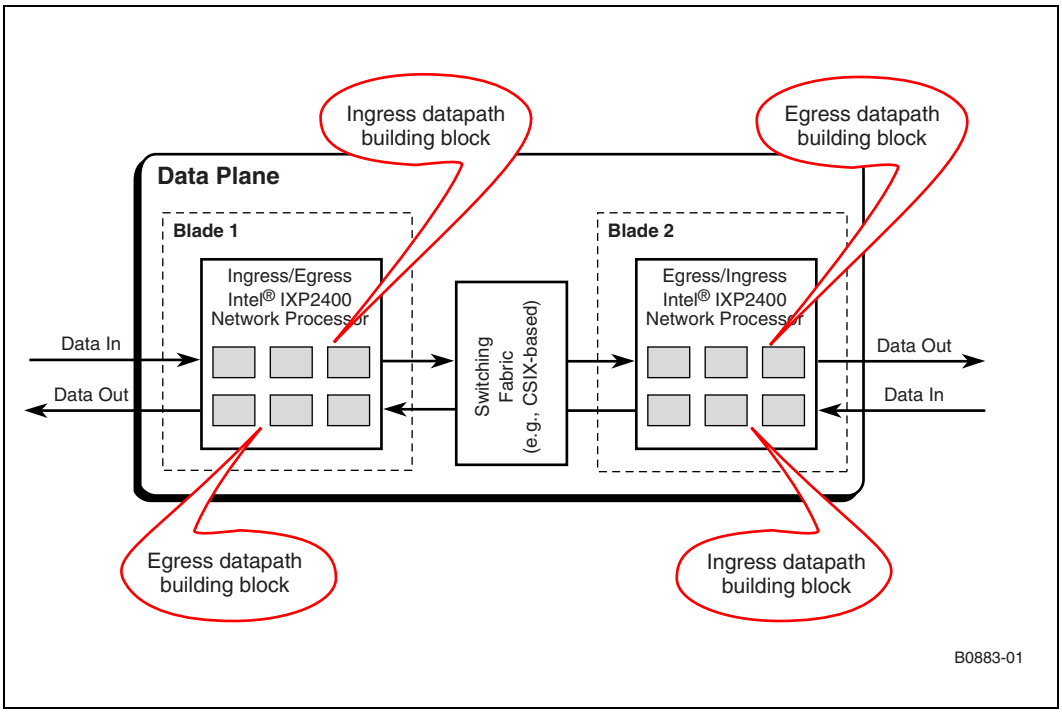


Figure 10-12 illustrates the single NP blade architecture, which may be possible in the future.

Figure 10-12. Single NP Blade Architecture

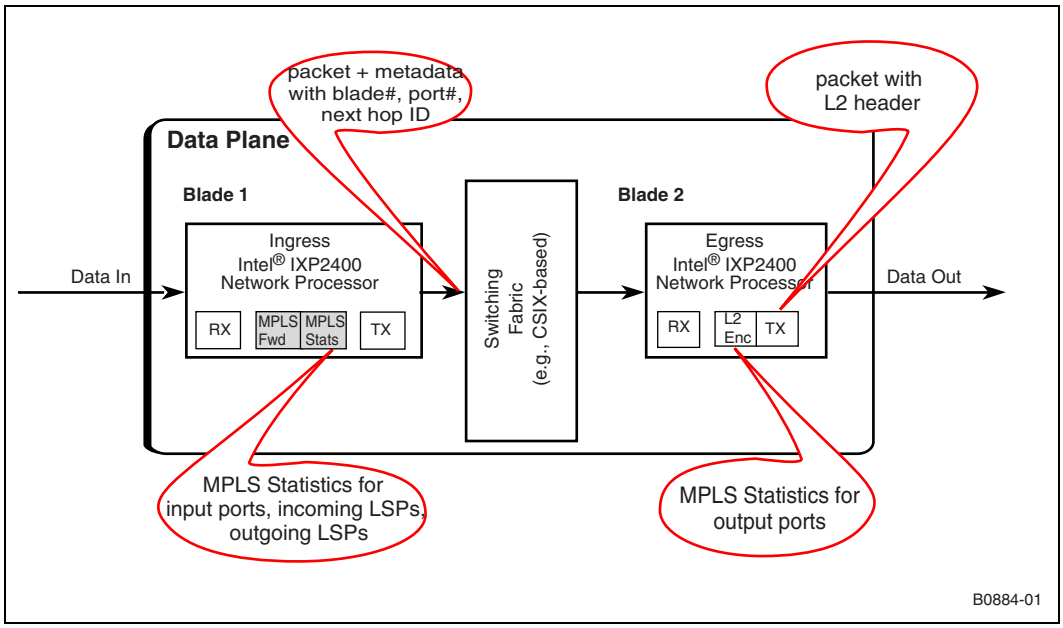


Because the data path through the system is the same as in the previous case (external interface 1 - backplane port 1 - backplane port 2 - external interface 2), the software block division remains unchanged—they just reside on the same processor, instead of on separate processors. Therefore, the MPLS forwarder building blocks described in this document shall be applicable to both architectures.

10.4.2 Ingress and Egress Microblocks

Figure 10-13 presents the high level view of the MPLS forwarding path with division between the ingress and egress Network Processor (this scenario applies both to the FTN and ILM Forwarders, and to single and dual IXP blade designs).

Figure 10-13. MPLS Ingress and Egress Microblocks



On the ingress NP, a data packet received from an external interface is stripped off of its L2 header by the interface RX microblock, and handed over to the MPLS forwarder (Fwd) block without the L2 encapsulation (the relevant L2 information is carried in a special meta-data structure associated with the packet). The MPLS forwarder processes the packet, determines its next hop, and stores the next hop information in the packet's meta-data. Additionally, the MPLS forwarder on the ingress NP gathers MPLS statistics connected with external input interfaces, incoming LSPs, and outgoing LSPs.

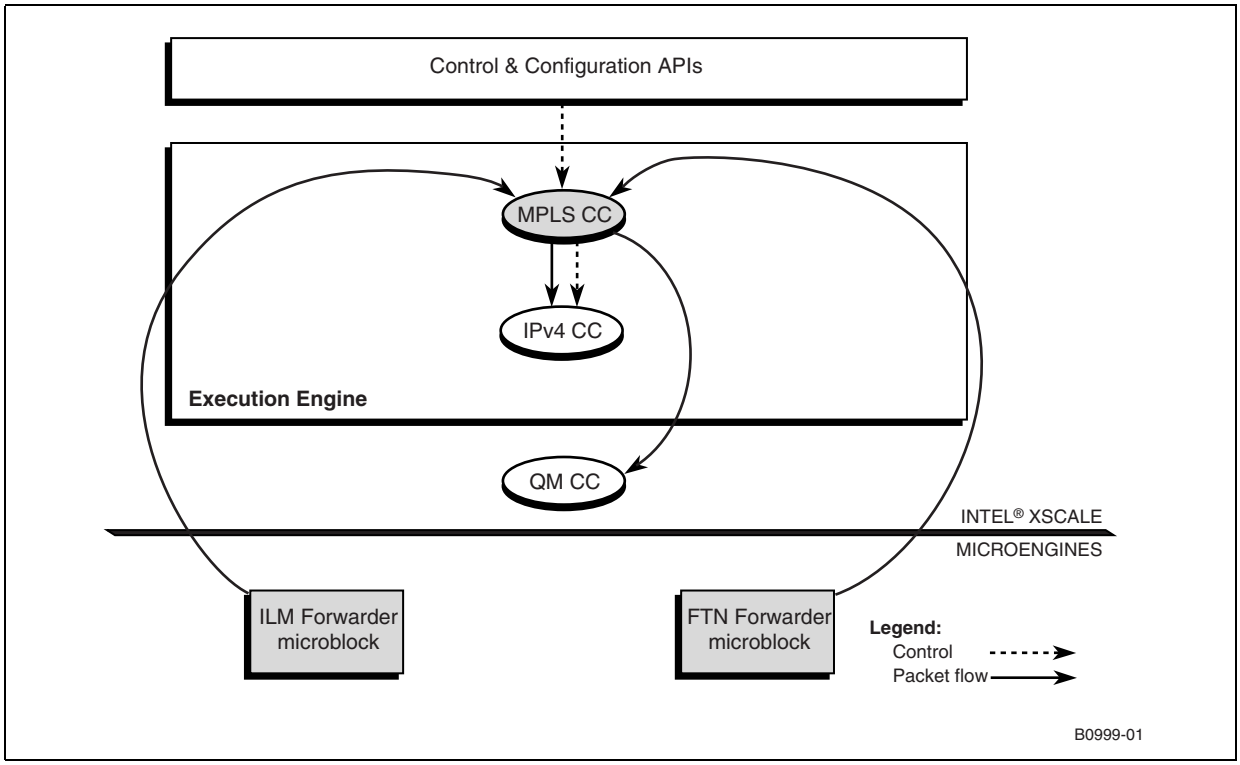
Additionally, MPLS statistics for external output interfaces is gathered by the L2 Encapsulation microblock on the egress NP. Such design prevents from splitting the MPLS forwarder microblock between ingress and egress NP. Then the packet is passed to the backplane TX microblock. The packet still does not contain the L2 encapsulation; its associated next hop information comprises the egress blade number, external output interface number and egress next hop ID. This information is transmitted in a special header together with the packet over the backplane to the egress NP. There it is processed by the L2 encapsulation (L2 Enc) part of the TX microblock. This block uses the egress next hop ID to find a predefined L2 header that should be applied to the packet before sending it through the output external interface. Predefined L2 headers for known IP next hops are created by L2/ARP core components in a way described in the *Intel® Internet Exchange Architecture Software Building Blocks Developer's Manual*, Chapter 47, "Ethernet ARP Module," and Chapter 62, "L2 Table Manager.". To be applicable to MPLS frames, the hardware protocol type in the predefined L2 header must be set to a media-specific value.

The division of the data path between media-independent forwarding and L2 encapsulation simplifies implementation, and allows the L2 encapsulation and transmit microblock to be reused by IP and MPLS forwarders.

10.4.3 MPLS Forwarder Core Component Overview

Figure 10-14 illustrates the MPLS core component and their interactions with other core components in the system. The dashed arrows in Figure 10-14 denote control messages or function calls, whereas the solid arrows show packet flows between components..

Figure 10-14. MPLS Forwarder Core Component



The MPLS Core Component operates on behalf both the ILM Forwarder and FTN Forwarder microblocks. It creates and maintains data structures shared among the microblocks and the core component, and the data comprises:

- MPLS statistics counters,
- NHLFE table,
- ILM table.

During initialization, the MPLS Core Component configures the ILM Forwarder and FTN Forwarder microblocks, patching it with the ILM, NHLFE, and MPLS statistics table base addresses.

During normal operation, the MPLS Core Component performs two kinds of tasks:

- Processes ILM and NHLFE table add/delete requests issued by the control plane,
- Processes MPLS exception packets sent to it by the MPLS microblocks.

A FEC to NHLFE mapping add/delete request comprises a FEC definition (in most cases, an IP address and mask pair), and a set of corresponding NHLFE table entries. Therefore, it involves changes in both the IP Routing table and NHLFE table. The MPLS core component adds or deletes MPLS FECs to the IP Routing table by means of the IPv4 Core Component, and updates the NHLFE table. The NHLFE table is used by the FTN Forwarder microblock.

An LSP add/delete request comprises an MPLS label, and a set of corresponding NHLFE table entries. Due to performance requirements, the ILM and NHLFE information for a given LSP is combined into one ILM table entry, independent from the NHLFE table used by the FTN Forwarder. The ILM table is used by the ILM Forwarder microblock.

The ILM Forwarder microblock sends exception packets to the MPLS Core Component if:

- MPLS-labeled packet's length exceeds the maxLabPktSize parameter value set for the output port. Depending on the DF bit value from the packet's IP header, it can be fragmented and forwarded as several shorter MPLS packets through the MPLS core component (slow path), or an ICMP error message can be generated. If the exception packet is a non-IP packet, it is dropped.
- If the packet's Time-to-live (TTL) value is not greater than the TTL value from the appropriate ILM table entry (typically 1). In this case an ICMP "Time exceeded" message is generated, encapsulated in the packet's label stack and forwarded towards packet's destination.
- MPLS packet with the “Router Alert” label has been detected (label value 1). In this case, the actual forwarding of the packet depends on the label beneath the "Router Alert" label - the packet may be forwarded as one of the following:
- MPLS packet through the MPLS Core Component (slow path),
- IP packet through the IPv4 Core Component,
- Generated an ICMP error message.

The FTN Forwarder microblock sends exception packets to the MPLS Core Component if:

- MPLS-labeled packet's length exceeds the maxLabPktSize parameter value set for the output port. Depending on the DF bit value from the packet's IP header, it can be fragmented and forwarded as several shorter MPLS packets through the MPLS core component (slow path), or an ICMP error message can be generated. If the exception packet is a non-IP packet, it is dropped.
- If the packet's TTL value is not greater than the TTL value from the appropriate FTN table entry (typically 1). In this case an ICMP “Time exceeded” message is generated, encapsulated in the packet's label stack and forwarded towards packet's destination.

10.4.3.1 Inter-Component Dependencies

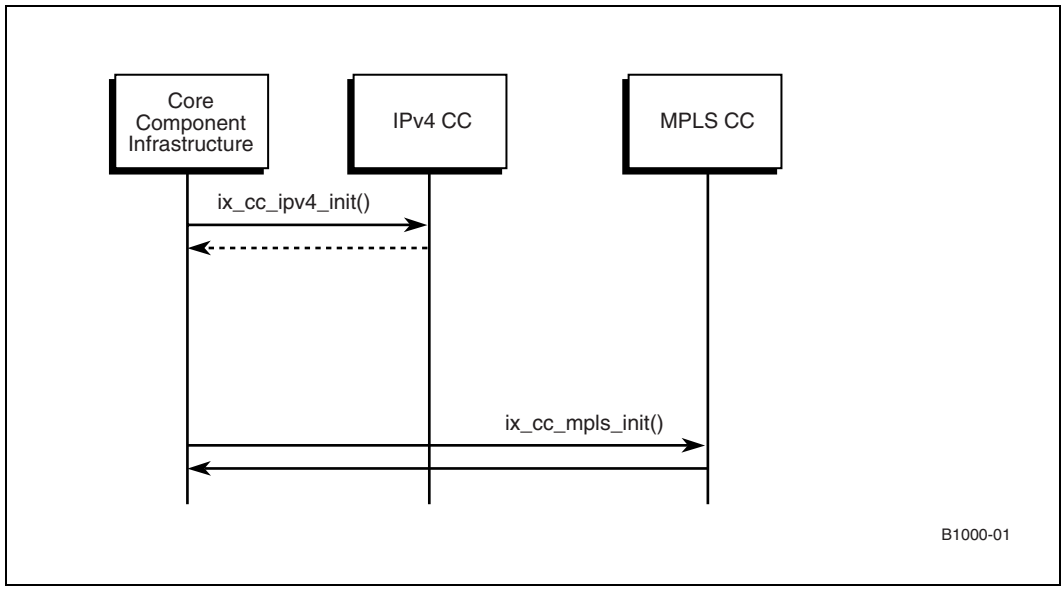
10.4.3.1.1 Operational Environment

The MPLS core component operates in the environment defined by the Core Component Infrastructure. This infrastructure allocates individual core components to so called execution engines (threads). To avoid synchronization problems in accessing shared data structures, the MPLS core component should be placed in the same execution engine as the IP forwarder core component, because it assumes sharing the IP Routing table with the IPv4 Forwarder. Figure 10-14 illustrates the indicated dependencies by placing the relevant components inside a rectangle labeled "Execution Engine".

10.4.3.1.2 Initialization Order

The MPLS core component should be initialized after the IPv4 Core Component, because it shares the IP Routing table with this component. Figure 10-15 illustrates the required initialization order..

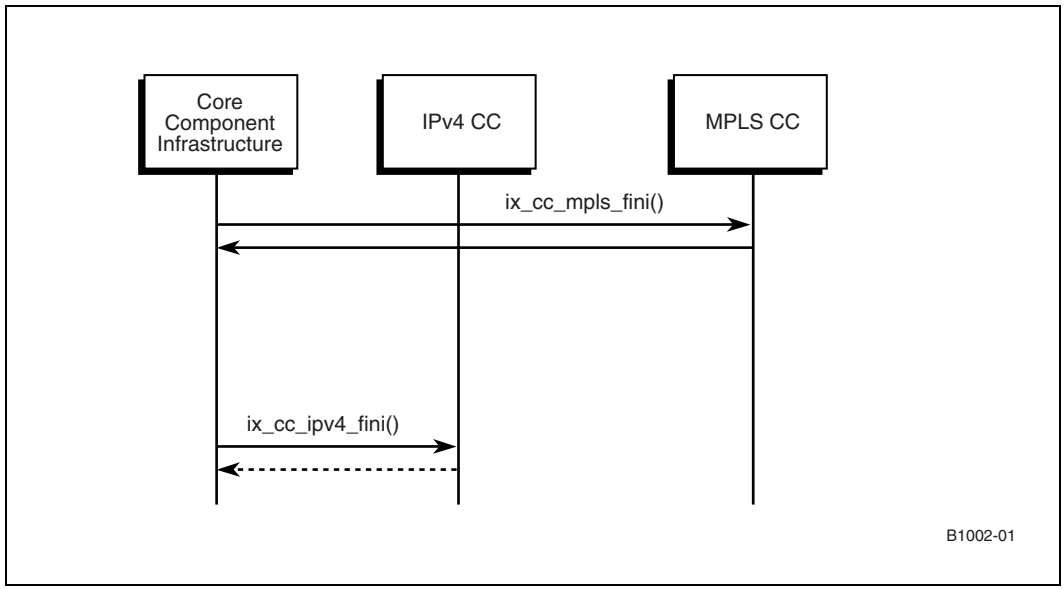
Figure 10-15. MPLS Core Component Initialization Order



10.4.3.1.3 De-Initialization Order

The MPLS Core Component de-initialization should be performed in the reverse order to its initialization sequence. Because the MPLS Core Component shares the IP forwarding table with the IPv4 Core Component, it is required that the IPv4 Core Component be de-initialized after the MPLS Core Component. Figure 10-16 illustrates the MPLS Core Component de-initialization order..

Figure 10-16. MPLS Core Component De-Initialization Order



10.4.3.1.4 Guaranteed Shared Data Validity

The MPLS core component must ensure that other core components and the MPLS microblocks always have access to valid shared data. This is achieved by utilizing two general techniques:

- Shared data validity among the MPLS core component and other core components is guaranteed by placing the affected core components in the same execution engine (thread). In this way it is possible to implement atomic shared data updates without the need for table locking.
- Integrity of the forwarding tables and other data shared among the MPLS core component and microengines is achieved by adding and removing data in a specific order. Each table entry has a flag indicating whether this entry is valid. In a multi-word entry, an update operation is performed by first invalidating the entry with an atomic memory swap operation, then setting all other data to the desired values, and finally validating the entry with another atomic memory swap. In a set of related entries, an update operation is performed by first invalidating the first (master) entry in the lookup sequence, then changing all entries pointed to by the master entry, and finally validating the master entry with an atomic memory swap.

10 Gb Ethernet IPv4/IPv6 Forwarding/ Tunneling Application 11

This chapter describes the design of an 10Gb IPv4/IPv6 Forwarding and Tunneling application using the Intel® IXP2800 Network Processor. Two half-duplex IXP2800 network processors are used to implement a 10Gb (10x1Gb and 1x10Gb) Ethernet line card that interfaces to a CSIX switch fabric. This section provides a high-level design overview and lists the different software components used to build this application. It focuses only on the fast path or microengine components of the design. The Intel XScale® core components for this application are described in *Intel® Internet Exchange Architecture Portability Framework Reference Manual*.

The application described in this chapter is supported on the Intel® IXDP2800 Advanced Development Platform.

Note: The designs for 10x1Gb and 1x10Gb applications are almost identical except for the packet Tx blocks. Such exceptions are explicitly noted in this chapter. All other details are applicable to both applications.

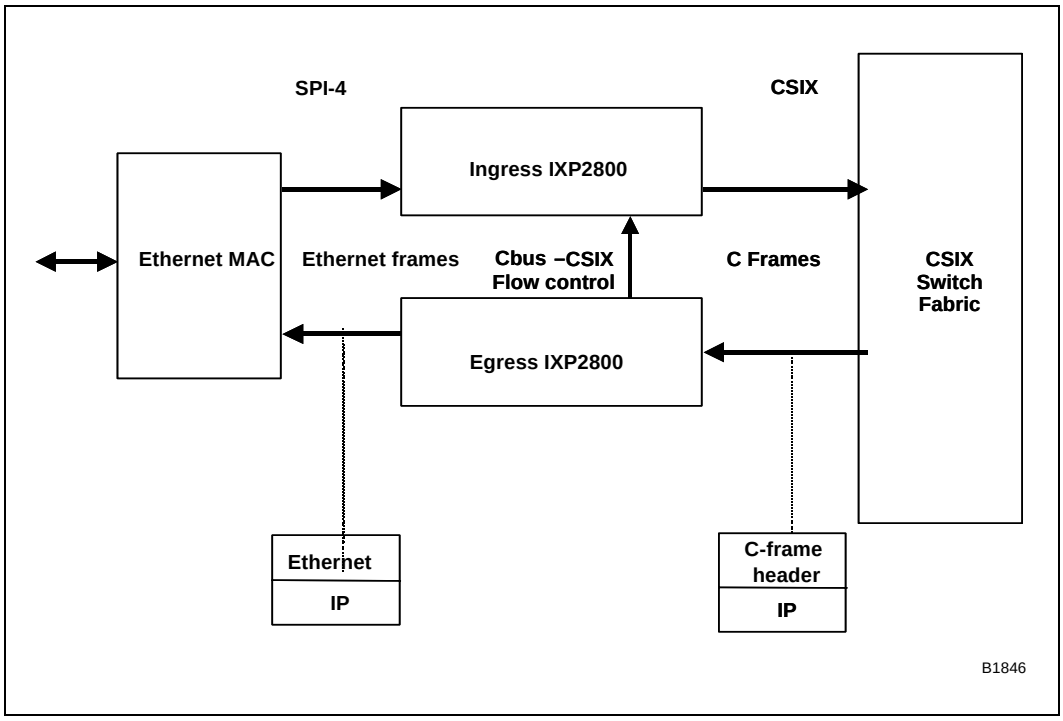
11.1 Hardware Overview

Figure 11-1 shows two Intel® IXP2800 Network Processors in a typical CSIX full duplex configuration. In this configuration, the two IXP2800 processors are identified as the ingress processor (receives from the Media interface and transmits to the CSIX Fabric) and the egress processor (receives from the CSIX Fabric and transmits to the Media interface). The hardware is configured in SPI-4 mode. Up to 10 Gigabit Ethernet ports are supported.

The Ingress IXP2800 receives Ethernet frames that carry IPv4 datagrams. The frames are assembled into IPv4 packets and the Layer-2 (Ethernet) headers are removed. Based on the IPv4 header, a Longest Prefix Match (LPM) lookup is performed and the packets are segmented into CSIX C-Frames and transmitted to the CSIX fabric. The result of the LPM lookup determines which IXP2800 connected to the Fabric receives the packet, and on which port on that IXP2800 the packet is transmitted.

The Egress IXP2800 receives CSIX C-Frames from the fabric and reassembles these into IPv4 datagrams. The Layer-2 (Ethernet) headers are added and the packets are transmitted over the appropriate port.

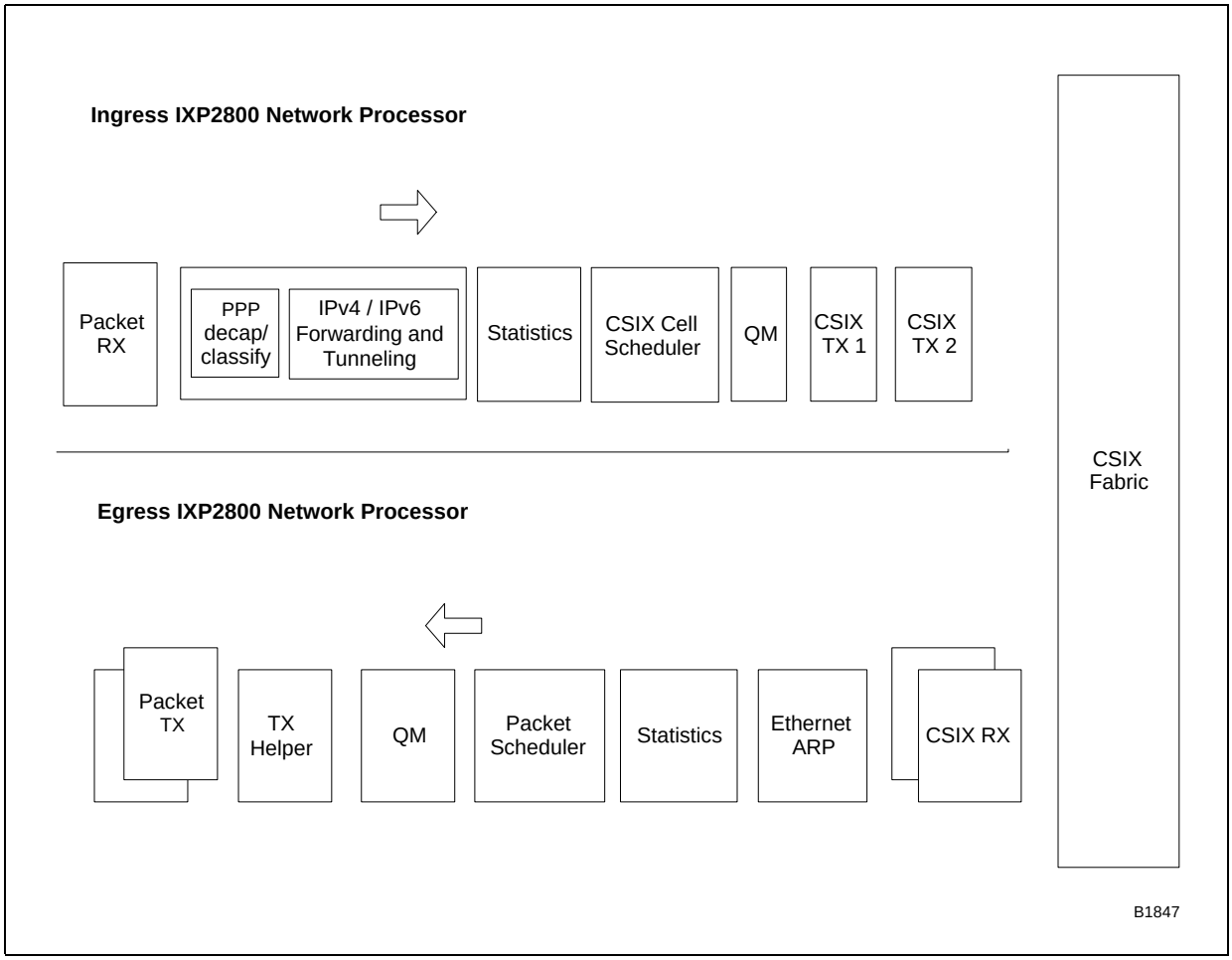
Figure 11-1. Example Hardware Configuration for 10x1/1x10 Gb Ethernet with CSIX Fabric



11.2 Software Overview

Figure 11-2 shows the microblocks needed to implement an Ethernet 10x1 Gb or 1x10 Gb IPv4/ IPv6 forwarding/tunneling application. The design for this application is based on the guidelines specified in the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*. The driver microblocks (Receive, Transmit, Scheduler and QM) run on different microengines to process the packets.

Figure 11-2. Microblocks for an Ethernet 10x1/1x10 Gb IPv4 Forwarding Application



11.2.1 Data Flow for the Ingress IXP2800

The following sections describe the data flow on the ingress IXP2800 network processors.

11.2.1.1 Packet RX

The packet RX block is identical to the Packet RX block described in Section 5.2.1.1, "Packet RX" on page 67 in Chapter 5, "OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application" except that it sets the header type field in the packet meta data to Ethernet and runs on one microengine (ME).

11.2.1.2 Packet Processing Microengines (PPP Decap/Classify + IPv4/IPv6 Forwarding + Tunneling)

The Packet Processing microengines (Ethernet Decap/Classify + IPv4/v6 Forwarding + Tunneling) are identical to the Packet processing RX block in Section 5.2.1.2, "Packet Processing Microengines (PPP Decap/Classify + IPv4/IPv6 Forwarder/Tunneling)" on page 68 of Chapter 5, "OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application", except that the Ethernet decapsulation/classify/filter module replaces the PPP decapsulation/classify module.

The Ethernet decapsulation/classify/filter module removes the layer-2 Ethernet header from the packet by updating the offset and size fields in the packet descriptor. It also implements MAC filtering based on the destination MAC address in the Ethernet header. Based on this filtering, the packet may be dropped.

11.2.1.3 Statistics Microblock

This block is identical to the Statistics block described in Section 5.2.1.3, "Statistics Microblock" on page 69 of Chapter 5, "OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application".

11.2.1.4 CSIX Scheduler

This block is identical to the CSIX Scheduler block described in Section 5.2.1.4, "CSIX Scheduler" on page 69 of Chapter 5, "OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application".

11.2.1.5 Cell Based Queue Manager (Cell QM)

This block is identical to the Cell Based Queue Manager block described in Section 5.2.1.5, "Cell Based Queue Manager (Cell QM)" on page 70 of Chapter 5, "OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application".

11.2.1.6 CSIX TX

This block is identical to the CSIX TX block described in Section 5.2.1.6, "CSIX TX" on page 70 of Chapter 5, "OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application".

11.2.2 Data Flow for the Egress IXP2800

This section describes the data flow for the Egress IXP2800.

11.2.2.1 CSIX RX

This block is identical to the CSIX RX block described in Section 5.2.2.1, "CSIX RX" on page 71 of Chapter 5, "OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application" except it also puts output port information into the message it forwards to downstream microblocks to be used by Packet Scheduler.

11.2.2.2 Ethernet ARP Microblock

This block checks whether the incoming packet has a valid L2 header table entry based on next hop id in the meta data. If this entry is invalid, this packet is enqueued to be processed by the Intel XScale® core. This block also receives packets from XScale core and sends it to the next stage of the pipeline.

11.2.2.3 Statistics Microblock

This block runs on a single microengine. It is currently a place-holder for statistics handling. It is anticipated that when this application is extended for MPLS and DiffServ, this microblock is used to manage per-flow statistics.

It handles dropping of large packets that are stored in multiple buffers. It interfaces with Ethernet ARP block through scratch ring, and interfaces with Egress Packet Scheduler via the Next Neighbor ring.

11.2.2.4 Egress Packet Scheduler

This block is identical to the Egress Packet Scheduler described in Section 5.2.2.2, “Egress Packet Scheduler” on page 71 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application”.

Note: This scheduler is currently fully tested only in simulation mode. In the future release it will be tested on hardware. Currently we use a simple round robin scheduler when running this application on hardware.

11.2.2.5 Packet Based Queue Manager (Packet QM)

This block is identical to the Packet QM described in Section 5.2.2.3, “Packet Based Queue Manager (Packet QM)” on page 72 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application”.

11.2.2.6 TX Helper

This block helps the Packet TX block by performing the following two functions:

- Gets TX requests from the Packet QM block via the Next Neighbor ring and then forwards the TX request to the Packet TX block through the scratch ring.
- Updates the per-class counters in SRAM. These counters keep tracks of the number of packets transmitted per class for the DRR Egress Packet Scheduler. To do this, the TX Helper block reads packet meta data to find the class ID for each packet. Then it calculates the SRAM address of the counter, reads the counter, increments the content, and writes back the new value.

11.2.2.7 Packet TX

The Packet Transmit microblock transmits packets over the Ethernet media interface. There are two designs depending on whether the application is using 10x1 Gb Ethernet ports or 1x10 Gb Ethernet ports.

- For 10x1 Gb Ethernet ports, Packet TX runs on two microengines in a functional pipeline, each microengine handling the transmission of 5 ports, as described in the *Intel® Internet Exchange Architecture Software Building Blocks Developer's Manual*, Chapter 12, "Packet TX–Multiports Microblock."
- For 1x10 Gb Ethernet ports, Packet TX runs on two microengines in a context pipeline connected by a Next Neighbor ring, as described in Section 5.2.2.5, "Packet TX" on page 72 of Chapter 5, "OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application".

Packet Tx segments a packet into mpackets, and moves them into TBUFS for the MSF state machine to transmit. It also adds the layer-2 Ethernet header to the packet.

For optimum utilization of TBUF elements, all ports share the same segment of TBUF. The port status and port TX FIFO high/low watermark is used to implement a flow control mechanism to prevent head of line blocking between ports.

Before the Packet TX moves data into TBUF, it makes sure that the TBUF in-flight (data filled, but not transmitted out of TBUF) does not exceed a predefined threshold to prevent TBUF overwriting.

This block also periodically updates the scheduler with information about how many packets have been transmitted. If the packets in flight for a particular port (packets scheduled but not transmitted) exceed a certain limit (which depends on the bandwidth supported by that port), then the scheduler stops scheduling any more packets for the port.

11.3 Performance Characterization

The Intel® IXP2800 Network Processor operates at 1400 MHz. For a minimum Ethernet packet of 64B, the packet inter-arrival time at 10 Gbps line rate is 94 ME cycles. If only one microgine is performing a specific function in the pipeline, then in order to maintain line rate for minimum packets, that microblock of the pipeline needs to retire a packet every 94 cycles. If *n* microengines are sharing a specific function in the pipeline, then that microblock needs to retire a minimum packet every *n*94* cycles. For example, there are two microengines sharing the transmission handling in Packet TX, then the budget for the min packet is `2*94 =188 cycles`.

Table 11-1 summarizes the performance analysis for the Ethernet pipeline.

Table 11-1. Summary of Performance Analysis for the Ethernet Pipeline

Line rate for 10 Gig Ports	10 Gigabits/sec
Min Ethernet packet size	64 bytes (+ 20 byte inter packet gap)
Packet Throughput for min packets	14.88 million packets/sec = (10 / (84*8)) * (10**9)
IXP2800 clock frequency	1400 MHZ
Inter-packet arrival time for min packets	1400/14.88 = 94 cycles
Compute cycles per packet for a single microengine	94
Latency per packet for a single microengine	94 * 8
Compute cycles per packet for n microengines running in parallel	94*n
Latency per packet for n microengines running in parallel	94*8*n

11.4 Ingress System Resource Allocation

The tables 11-2 and 11-3 show the system resources mapped for the Ingress IXP2800. This mapping reflects the system defaults that may be changed. The allocation of microengines is done such that it optimizes the performance of this specific application and may be changed for other applications.

Table 11-2. System Resources Mapped for the Ingress IXP2800

Microblock	ME #	Communication Mechanism with previous stage
Packet RX	ME 1:2	Auto-push status from MSF
IPv4 Forwarder + Layer2 decapsulation/Classify	ME 0:0, 0:1, 0:2, 0:3, 0:4, 1:4, 1:5	Scratch ring
Statistics	ME 0:5	Scratch ring
CSIX Scheduler	ME 0:6	NN ring
Queue Manager	ME 0:7	NN ring
CSIX TX	ME 1:0, 1:1	NN ring
Headroom	3 microengines	

The physical assignment of function to microengine is important since it not only affects when the next neighbor registers and signaling can be utilized, but it also affects the utilization of the internal Command bus and S-Push/Pull buses. This assignment attempts to balance the usage of the Command bus and S-Push/Pull buses across the two clusters.

Note: These values are defined in a system header file `dl_system.h` and may be changed as required.

The IXP2800 supports four SRAM channels and three DRAM channel. Table 11-3 shows the SRAM, DRAM and scratch utilization for the 10GB Ethernet IPv4/IPv6 Forwarding/Tunneling Application.

Table 11-3. SRAM, DRAM and Scratch Utilized for Ingress IXP2800

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Buffer Descriptors	32	32k (In simulation, we use only 320 buffers)	1 MB		
Buffers	2048	32k		64 MB	
Queue Descriptors	16	256 (1 per VOQ)	4K		
CSIX TX contexts	32	256 (1 per VOQ)	8k		
Trie Table	64 (The root Trie table requires at least 257k to support hi64k and hi256 tables. In addition each node requires 64 bytes. These nodes are added as needed)	See note in previous column. Assuming 256k routes, approximately 128k nodes are needed	8MB		
Route Table (Next Hop Information)	16	Assuming 4k next hops	64k		
IPv4 statistics	4	16			64

Table 11-3. SRAM, DRAM and Scratch Utilized for Ingress IXP2800 (Continued)

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Packet RX statistics	4	16*16	1024		
IPv4 Directed Broadcast Table	32 (local memory)	64			
Ring from Packet RX to packet processing pipeline (IPv4+Layer2 Decap/ Classify)	12	2k/12			2k
IPv4 to Statistics ring	12	2k/12			2k
QM Q-Array entries	N/A	16			
Buffer Free list Q-Array entry	N/A	4			

11.5 Egress System Resource Allocation

Table 11-4 shows the system resources allocation for the Egress IXP2800.

Table 11-4. System Resources Allocation for the Egress IXP2800

Microblock	ME #	Communication Mechanism with previous stage
CSIX RX	ME 1:1, 1:3	Auto-push status from MSF
Ethernet ARP	ME 0:1	Scratch ring
Statistics	ME 0:2	Scratch ring
Egress Scheduler	ME 0:3, 0:4, 0:5	NN ring
Egress QM	ME 0:6	NN ring
TX Helper	ME 0:7	NN ring
Packet TX	ME 1:0, 1:2	Scratch ring
Headroom	5 microengines	N/A

The mapping of networking functions on to the microengines shows that 11 microengines are used to perform the fast path processing for this application. Additional functionality required by customers can be mapped on to the remaining microengines.

Table 11-5 shows the SRAM, DRAM and scratch utilization for the 10GB Ethernet IPv4/IPv6 Forwarding/Tunneling Application. These values are defined in a system header file dl_system.h and may be changed as needed.

Table 11-5. SRAM, DRAM and Scratch Utilization for Egress IXP2800

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM Used	Total Scratch used
Buffer Descriptors	32	32k (In simulation we use only 320 buffers)	1 MB		
Queue Descriptors	16	256 (16 ports x 16 classes per port)	4k		
CSIX RX Reassembly contexts	32	1024	32k		
Buffers	2048	32k		64 MB	
CSIX RX to Ethernet ARP ring	12	2k/12 (the size of the ring is 512 long words, but each entry enqueued uses 3 long words. Therefore the total number of entries is 512/3 = 170)			2k
Ethernet ARP ring to Statistics Ring	12	2k/12 (the size of the ring is 512 long words, but each entry enqueued uses 3 long words. Therefore the total number of entries is 512/3 = 170)			2k
Inside Scheduler between Count block and Class Scheduler block	4	512			2k
Layer 2 table with mapping from next hop id to Ethernet header	16	65536	1 MB		
TX Helper to first Packet TX	4	256			1K
TX Helper to second Packet TX	4	256			1K
QM Q-Array entries	N/A	16			
Buffer Free list Q-Array entry	N/A	4			

11.6 Interfaces Between the Various Microblocks

This section describes the interfaces between the different microblocks in the ingress and egress processors for this application.

11.6.1 Packet RX and Packet Processing Microengines

The interface between the Packet RX microblock and the packet processing microengines is a scratch ring. Table 11-6 describes each entry in the scratch ring—which is three long words.

The format depends on whether the packet fits in one buffer or not. In the case of packets that span across multiple buffers, some of the packet descriptor information is written to SRAM and the rest to the scratch ring. In the case of packets that fit into a single buffer, all the information is packed into the scratch ring eliminating one read/write to SRAM in the critical path. Bit 31 of LW0 (EOP bit of the handle) is used to detect if a packet spans across multiple buffers. If this bit is set (implying that the buffer is a SOP/EOP buffer), then the packet is contained in a single buffer.

This interface is used for packets that fit entirely in one buffer.

Table 11-6. Three-Word Scratch Ring Entry —Packets fit on one Buffer

LW	Bits	Size	Field	Description
0	31:0	32	dl_buffer_handle	Buffer Handle for the SOP Descriptor
1	31:16	16	input_port	Input port on ingress processor
	15:12	4	free_list_id	Free list ID for buffer
	11:8	4	rx_stat	Receive Status Flag
	7:0	8	header_type	Type of header at offset bytes into the packet
2	31:16	16	buffer_size	Buffer size in bytes
	15:0	16	offset	Offset of the start of data in the SOP buffer in bytes

This interface is used for packets that require more than one buffer.

Table 11-7. Three-Word Scratch Ring Entry —Packets Require more than one Buffer

LW	Bits	Size	Field	Description
0	31:0	32	dl_buffer_handle	Buffer Handle for the SOP Descriptor
1	32:0	32	dl_eop_buffer_handle	Buffer Handle for the EOP Descriptor
2	31:16	16	packet_size	Total packet size across buffers in bytes
	15:0	16	offset	Offset of the start of data in the SOP buffer in bytes

11.6.2 Packet Processing Microengines and Statistics

The Packet Processing Microengines and Statistics interface is a scratch ring. Table 11-8 describes each entry in the scratch ring—which is three long words.

Table 11-8. Three-Word Scratch Ring Entry—Packet Processing Microengines and Statistics

LW	Bits	Size	Field	Description
0	30:16	16	MOP_EOP_buf_size	Size in bytes of all MOP buffers and the EOP buffer of the packet
0	0:15	16	Queue Number	Queue Number
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

11.6.3 Statistics and CSIX Scheduler

The Statistics and CSIX Scheduler interface is a next neighbor (NN) ring. Table 11-9 describes each entry in the NN ring—which is three long words.

Table 11-9. Three-Word NN Ring Entry (Statistics and CSIX Scheduler)

LW	Bits	Size	Field	Description
0	30:16	16	Packet cell count	Sum of all buffer cell counts belonging to the packet
0	0:15	16	Queue Number	Queue Number
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

11.6.4 CSIX Scheduler and Cell Queue Manager

The CSIX Scheduler and Cell Queue Manager interface is a next neighbor ring. Table 11-10 describes each entry in the NN ring—which is three long words.

Table 11-10. Three-Word NN Ring Entry (CSIX Scheduler and Cell Queue Manager)

LW	Bits	Size	Field	Description
0	30:16	16	Dequeue Queue #	Queue number from which to dequeue. Zero implies no dequeue
0	0:15	16	Enqueue Queue #	Queue number on which to enqueue. Zero implies no enqueue
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

11.6.5 Cell Queue Manager and CSIX TX

The Cell Queue Manager and CSIX TX interface is a next neighbor ring. CSIX Transmit is a two-microengine context pipe-stage. The cell queue manager writes to the NN ring of the first CSIX TX microengine. Table 11-11 describes each entry in the NN ring—which is two words.

Table 11-11. Two-Word NN Ring Entry (Cell Queue Manager and CSIX TX)

LW	Bits	Size	Field	Description
0	31:16	16	Reserved	Reserved
0	15:0	16	Queue Number	Queue Number
1	31:0	32	Buffer Handle	Buffer Handle currently being transmitted for queue

11.6.6 CSIX TX—First ME to Second ME

The interface between the first CSIX TX microengine and second CSIX TX microengine is a next neighbor ring. Table 11-12 describes each entry in the NN ring—which is eight long words.

Table 11-12. Eight-Word NN Ring Entry (CSIX TX—First ME to Second ME)

LW	Bits	Size	Field	Description
0	31:0	32	Tx_request0	Same as LW0 from Cell Queue Manager to CSIX TX
1	31:0	32	Tx_request1	Same as LW1 from Cell Queue Manager to CSIX TX
2	31:0	32	dram_handle	DRAM address where CSIX cell is stored
3	31:24	8	cell_count_remaining	Number of cells remaining in the current buffer
	23:18	6	Reserved	Reserved
	17:17	1	MOP_EOP_flag	If MOP_EOP, set to 1, else 0
	16:16	1	SOP_EOP_flag	If SOP and EOP, set to 0, else 1
	15:0	16	payload_length	Length of CSIX cell payload in bytes
4	31:0	32	prepend_header0	LW0 of CSIX cell pre-pend header
5	31:0	32	prepend_header1	LW1 of CSIX cell pre-pend header
6	31:0	32	prepend_header2	LW2 of CSIX cell pre-pend header
7	31:0	32	prepend_header3	LW3 of CSIX cell pre-pend header

11.6.7 CSIX RX and Ethernet ARP

The CSIX RX and Statistics interface is a scratch ring. Table 11-13 describes each entry in the scratch ring—which is three words

Table 11-13. Three-Word Scratch Ring Entry (CSIX RX and Statistics)

LW	Bits	Size	Field	Description
0	30:16	16	Packet Size	Packet Size
0	15:12	4	Port Number	Output Port Number
0	11:0	12	Queue Number	Queue Number
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

11.6.8 Ethernet ARP and Statistics

The Ethernet ARP to Statistics interface is a scratch ring. Table 11-14 describes each entry in the NN ring - which is 3 long words.

Table 11-14. Three-Word Scratch Ring Entry (Statistics and Ethernet ARP)

LW	Bits	Size	Field	Description
0	30:16	16	Packet Size	Packet Size
0	15:12	4	Port Number	Output Port Number
0	11:0	12	Queue Number	Queue Number
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

11.6.9 Statistics and Packet Scheduler

Table 11-15 shows the Statistics and Packet Scheduler interface, which is a Next Neighbor ring.

Table 11-15. Three-Word NN Ring Entry (Statistics and Packet Scheduler)

LW	Bits	Size	Field	Description
0	30:16	16	Reserved	Reserved
0	15:0	16	Packet Size	Packet Size
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:16	16	Port Number	Output Port Number
2	31:0	16	Queue Number	Queue Number

11.6.10 Packet Scheduler and Queue Manager

The interface between the Queue Manager and the Packet Scheduler is a Next Neighbor Ring. Table 11-16 describes each entry in the NN ring—which is three long words.

Table 11-16. Three-word NN Ring Entry (Queue Manager and Packet Scheduler)

LW	Bits	Size	Field	Description
0	30:16	16	Dequeue Queue #	Queue number from which to dequeue. Zero implies no dequeue
0	0:15	16	Enqueue Queue #	Queue number on which to enqueue. Zero implies no enqueue
1	31:0	32	SOP Buffer Handle	Buffer Handle for SOP Descriptor
1	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

11.6.11 Queue Manager and TX helper

The interface between the Queue Manager and the TX helper is a Next Neighbor ring. Table 11-17 describes each entry in the NN ring—which is one word:

Table 11-17. Two-Word NN Ring Entry (Queue Manager and Packet TX)

LW	Bits	Size	Description
0	31:4	28	Reserved
0	3:0	4	Port number
1	31:24	8	Reserved
1	23:0	24	Pointer to SOP buffer descriptor in SRAM in long words (Same as bits 0:23 of buffer handle)

11.6.12 TX Helper and Packet TX (10x1 GigE)

The interface between TX Helper and Packet TX is different for 10x1 and 1x10 applications. For 10x1, the interface between the TX Helper and the Packet Transmit is two scratch rings—one for the first Packet TX ME which handles the transmission of port 0 to 4, and one for the second Packet TX ME which handles the transmission of port 5 to 9. Table 11-18 shows each entry is one long word in a scratch ring.

Table 11-18. Two Scratch Ring Interface (TX Helper and Packet TX)—One Word

LW	Bits	Size	Description
0	31:31	1	Valid bit
	30:28	3	Reserved
	27:24	4	Port number
	23:0	24	Pointer to SOP buffer descriptor in SRAM in long words (Same as bits 0:23 of buffer handle)

11.6.13 TX Helper and Packet TX (1x10 GigE)

The interface between the TX helper and the Packet Transmit for 1x10 applications is a Next Neighbor ring. Table 11-19 describes each entry in the NN ring—which is one word.

Table 11-19. One-Word NN Ring Entry (Queue Manager and Packet TX)

LW	Bits	Size	Description
0	31:31	1	Valid bit
0	30:28	3	Reserved
0	27:24	4	Port number
0	23:0	24	Pointer to SOP buffer descriptor in SRAM in long words (Same as bits 0:23 of buffer handle)

11.6.14 Packet TX —First ME to Second ME (1x10 GigE)

The interface between the first microengine and second microengine of Packet Transmit for 1x10 application is a Next Neighbor ring. Table 11-20 describes each entry in the NN ring—which is three words.

Table 11-20. Three-Word NN Ring Entry (Packet TX—First ME to Second ME)

LW	Bits	Size	Description
0	31:0	32	Pointer to meta data (used to free buffer
1	31	1	Bit is clear if the m-packet is sop
	30	1	Bit is clear if the m-packet is eop
	29:0	29	Offset of payload to be transmitted
2	31:0	32	Payload size to be transmitted

If the m-packet is non-stop, then 3 more long words are included on the ring.

Table 11-21. Three-Word NN Ring Entry (for Non-stop m-packet)

LW	Bits	Size	Description
3	31:0	32	Bytes from previous buffer to be prepended to the current buffer
4	31:0	32	Exe_stat_flag: information about various condition flags
5	31:0	32	Partially created transmit control word

Core Router Application 12

This chapter describes the design of a Core Router application (IPv4+IPv6+MPLS+Diffserv at OC192 data rate) using three Intel® IXP2800 Network Processors with headroom for additional functionality. It provides a high-level design overview and lists the different software components used to build this application. It focuses only on the fast path or microengine components of the design. The Intel XScale® core components for this application will be described in a future release.

The application described in this chapter is supported on the transactor (network processor simulator) and on the Intel® IXDP 2800 Advanced Development Platform.

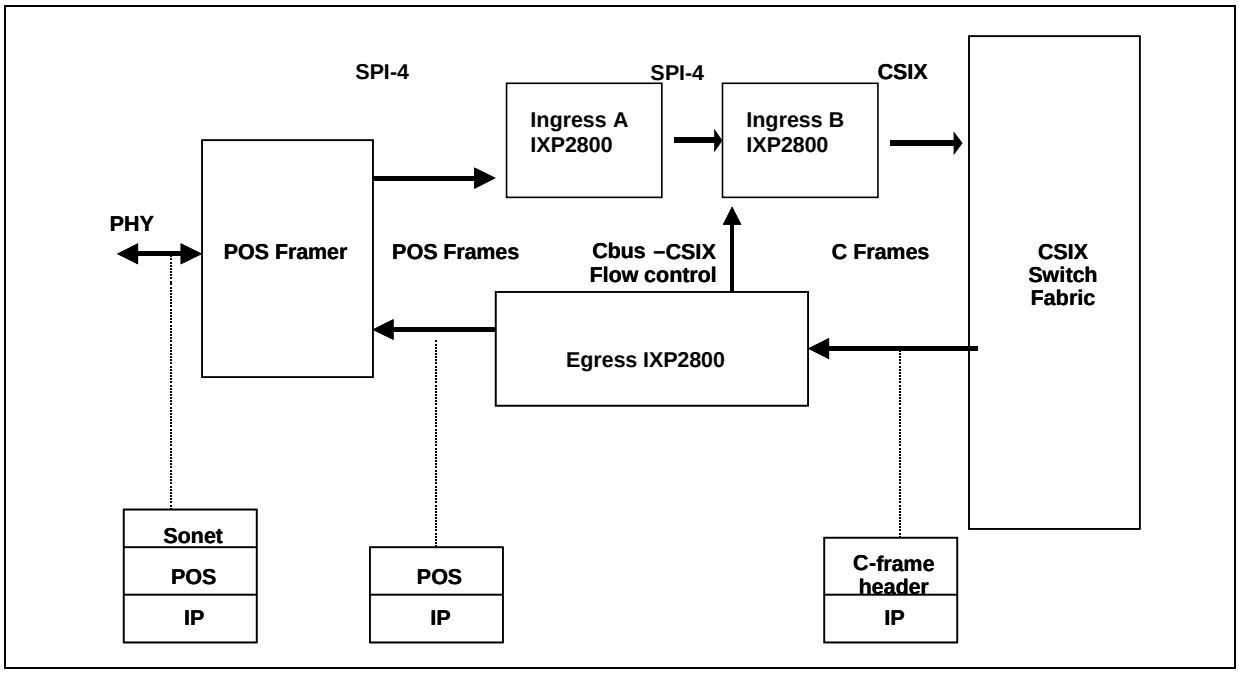
Note: The OC-192 POS IPv4/IPv6 Forwarding and Tunneling application and its associated microblocks are referred to frequently in this chapter. See Chapter 5, "OC-192 POS IPv4/IPv6 Forwarding/ Tunneling Application" for more details.

12.1 Hardware Overview

Figure 12-1 shows three Intel® IXP2800 Network Processors. In this configuration, the IXP2800s are identified as Ingress A, Ingress B and Egress IXP2800. A distinguishing feature is that Ingress A and Ingress B are cascaded serially through SPI4 to form a two-chip Ingress pipeline. Ingress A receives packets from the media interface, looks at the IPv4/IPv6/MPLS header and makes a forwarding decision, and transmits the packet over SPI4 to Ingress B. Ingress B receives packets

from Ingress A, sends it through Meter and WRED, and transmits the packet over CSIX Fabric to the appropriate egress blade. The Egress IXP2800 receives from CSIX Fabric and transmits to the media interface.

Figure 12-1. Example Hardware Configuration for Core Metro Application Using 3 IXP2800



12.2 Software Overview

Figure 12-2 shows the microblocks needed to implement the core router pipeline application described in this chapter. The design for this application is based on the guidelines specified by the IXA Portability Framework, which is described in the following user documents:

- *Intel® Internet Exchange Architecture Portability Framework Reference Manual*
- *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*

The driver microblocks (Receive, Transmit, Scheduler and QM) are similar to those used in the OC-192 POS IPv4/IPv6 forwarding and tunneling application, which is implemented on two IXP2800s and is described in Chapter 5, "OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application."

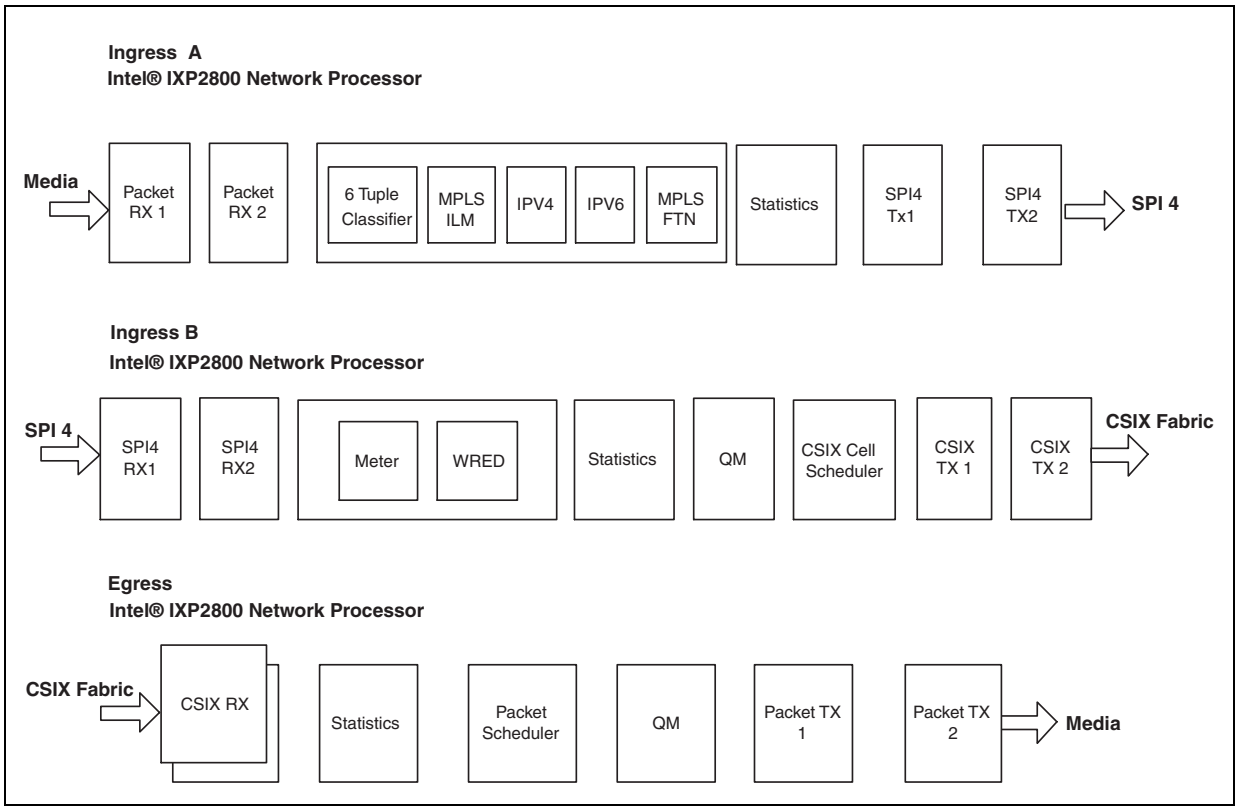
The packet processing microblocks (IPv4, IPv6 and MPLS) are similar to those used in other Intel® IXP2400 Network Processor applications and are reused with no modifications.

This application is made available in incremental steps:

- In its initial release, IPv6 and MPLS functionality is available in two different pipelines i.e IPv4/IPv6 pipeline and IPv4/MPLS pipeline because it needs larger control store (8k, available in IXP2800 B0).

- In a future release, the separate IPv6 and MPLS pipelines will be combined to form IPv4/IPv6/ MPLS pipeline. Also, DiffServ functionality, using 6 Tuple Classifier, Meter, and WRED microblocks, will be available in a future release.

Figure 12-2. Microblocks for Core Router Application



12.2.1 Data Flow for the Ingress A IXP2800

The following sections describe the data flow on the ingress A IXP2800 processor.

12.2.1.1 Packet RX

This block is identical to the Packet RX block described in Section 5.2.1.1, “Packet RX” on page 67 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application.”

12.2.1.2 Packet Processing Microengines

This section describes the following microengines:

- PPP decapsulation
- 6 tuple classifier
- IPv4 forwarder
- MPLS marker and switching
- IPv6 forwarder and tunneling

The PPP decapsulation microblock runs along with the IPv4/IPv6/MPLS microblocks on 8 microengines or 64 threads. The IPv4, IPV6 and MPLS microblocks are similar to the ones used in various IXP2400 applications and are completely reused.

An application specific system source microblock on each thread dequeues packet buffer handles from the scratch ring. This source block (DL_Source[]) is a system microblock implicit in the dispatch loop. It reads in the packet meta information (i.e. the packet descriptor) and populates the dispatch loop state. It also reads in 32 or 64 bytes of the packet header from DRAM into a header cache maintained in transfer registers (see Section 12.2.1.3 for details). Since it is important to maintain packet sequencing, the threads in the microblock execute in strict order to dequeue from the scratch ring. This implies that the first thread on microengine 1 dequeues the first packet, signals the next thread to perform the dequeue, etc. From this block, the packet goes to the PPP decapsulation/classify microblock.

The PPP decapsulation/classify microblock removes the layer-2 PPP header from the packet by updating the offset and size fields in the packet descriptor. Based on the PPP header, it also classifies the packet into IPv4, IPv6, MPLS, PPP control packet (LCP, IPCP etc). If the packet is a PPP control packet, it is marked as an exception packet to be sent to the XScale Core (IX_EXCEPTION). Otherwise the packet is sent down the microengine pipeline for further processing.

Depending on the packet type (IPv4. IPv6, MPLS) packets go through different microblocks. The following packet flow path will be described based on how each packet type is handled.

- IPv4 forwarding
- IPv6 forwarding
- IPv6 tunnel encapsulation and decapsulation
- Ingress LER
- LSR and egress LER

12.2.1.2.1 IPv4 Forwarding

The IPv4 forwarder microblock validates the IP header per RFC 1812. If the validity checks fail, then the packet is set up to be dropped as specified in the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*.

Otherwise a Longest Prefix Match (LPM) is performed on the IPv4 header. The result is an IPv4 Next Hop ID, a fabric blade id (identifying a unique IXP2800 on the fabric) and an output port identifying the output port on the egress IXP2800. All three fields are passed to Ingress B which in turn is sent to Egress processor where the information is used to appropriately queue and transmit the packet.

If no LPM match is found, then the packet is set up to be sent up to the XScale core for further processing as specified in [IXASF]. Packets are also sent to the core in a number of other cases, for example when the packet is destined for a local interface or is to be fragmented. (Since no core components are available for this application the packets will simply be dropped in such cases).

IPv6 packets are handled by three microblocks: IPv6 Forwarder, Tunnel Decap and Tunnel Encap microblocks.

12.2.1.2.2 IPv6 Forwarding

IPv6 Forwarder validates IPv6 Header and address per RFC 2460 and 2373 respectively. If the check fails, then the packet is set up to be dropped as specified in the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*. Otherwise a Longest Prefix Match (LPM) is performed on IPv6 destination address. It uses 16, 8,…8 bit prefixes to match 128 bit IPv6 address. The result is an IPv6 Nexthop ID, a fabric blade id (identifying a unique IXP2800 on the fabric) and an output port identifying the output port on Egress IXP2800. All three fields are sent over to Ingress B which in turn is sent over to Egress where it is used to dispose the packet appropriately.

12.2.1.2.3 IPv6 over IPv4 Tunneling

The tunneling microblocks provide the capability for the node to serve as an endpoint of an IPv6 over IPv4 tunnel. The types of tunneling supported are:

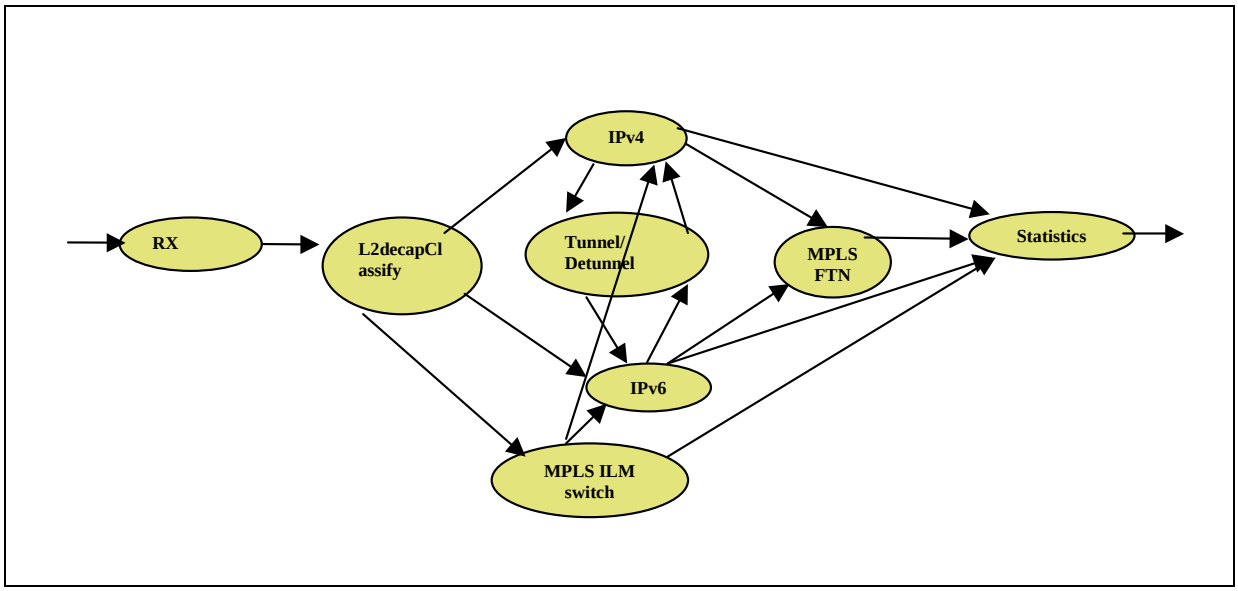
- Configured tunnels as defined in RFC 2893
- Automatic tunnels as defined in RFC 2893
- 6to4 tunnels as defined in RFC 3056

The V6V4-Tunnel-Decap microblock handles IPv4 packets that contain an encapsulated IPv6 packet and that have reached the tunnel endpoint. The V6V4-Tunnel-Encap microblock handles IPv6 packets that require encapsulation in an IPv4 packet in order to reach the next hop IPv6 node (passing through one or more IPv4 only node).

A tunnelled IPv6 (over IPv4) packet is first processed by IPv4 forwarder microblock. The nexthop ID obtained after an LPM lookup identifies a tunnel end point and the packet is passed to Tunnel Decap microblock. The encapsulating IPv4 header is then removed by Tunnel Decap and the exposed IPv6 packet is sent to IPv6 microblock for forwarding. After IPv6 LPM lookup, the nexthop ID obtained identifies whether this IPv6 packet goes through a tunnel or not. If a tunnel is required, then the packet is sent to Tunnel Encap microblock where an IPv4 header is added to the IPv6 packet. The resulting IPv4 packet is forwarded by IPv4 Forwarder microblock just like any other IPv4 packet.

Figure 12-3 shows the packet flow for the Ingress A processor in this pipeline application. In implementation, this flow will be converted to a flattened Microblock Call graph as explained in the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*, in the Dispatch Loop chapter.

Figure 12-3. Packet Flow in Ingress A



12.2.1.2.4 Ingress LER

In the case of an Ingress LER, an incoming IPv4 (IPv6) packet is first processed by an IPv4 (IPv6) microblock (as previously described in Section 12.2.1.2.1 and Section 12.2.1.2.2). The nexthop ID obtained indicates if this packet goes through an MPLS LSP or not. If yes, further processing is done by MPLS microblock. Using the nexthop ID as an index into FTN (FEC to NHLFE) table, information required to form MPLS header such as the number of labels to be pushed (max of 4 PUSH/POP supported) and the label value for each push, is obtained. The MPLS header thus formed is prepended to IPv4 packet and sent over to Ingress B. The case of Ingress LER handling IPv6 packets is similar except the packet is first processed by IPv6 microblock instead of IPv4.

Note: With IPv6 over IPv4 tunneling, more data paths are possible as shown in Figure 12-3, but the end result of tunelling is always an IPv4 or IPv6 packet which is then handled as explained in this section.

12.2.1.2.5 LSR and Egress LER

In the case of LSR or Egress LER, the incoming packet is an MPLS packet. Using the incoming label as an index into ILM (Incoming Label Map) table the nexthop ID is obtained, which provides info like the operation to perform (PUSH or POP), number of labels to PUSH, the label value for each PUSH operation etc. Based on this info the packet header is accordingly modified. For Egress LER (at IP/MPLS domain edge) the resulting packet will be an IPv4/IPv6 packet. Subsequent forwarding of this packet is done based on the resulting IPv4/IPv6 packet header by passing the packet through IPv4/IPv6 microblock. For LSR not at penultimate hop, the resulting packet will be a MPLS packet. For LSR at penultimate hop, the resulting packet will be an IPv4 or IPv6 packet but forwarding is based on the POPed label, so no further lookup is necessary for the resulting IPv4 packet.

The MPLS microblock supports both per interface label space as well as per platform label space. It also supports NHLFE sets which is useful in load balancing, fault tolerance and diffserv implementations.

At the end of packet processing stage, the packet is passed on to an application specific system microblock (DL_Sink[]). DL_Sink[] simply writes the modified packet header to DRAM and the packet meta information to SRAM and sends a message to the next microblock in the pipeline, that is, the statistics microblock.

12.2.1.3 Dispatch Loop / Microblock Groups

One of the challenges of implementing this packet processing stage is the dispatch loop that brings all microblocks together.

The packet header size varies from a minimum of 20 bytes (IPv4 only) to a maximum of 56 bytes (4 labels + IPv6 (40)). Reading in all 56 bytes for every packet wastes DRAM bandwidth that then impacts performance of 40B IPv4 min packets (24.5 mpps). Reading in only 20 or 32 bytes of header results in additional reads for IPv6 and MPLS POP3/4 operations resulting in performance impact for those cases.

This is solved by taking advantage of the fact that larger the header size (and hence packet size), the lower the packet rate (40B IPv4 is 24.5 mpps. 60B v6 is 16.3 mpps), and hence more memory bandwidth (and instruction budget) for larger packets. So, for example, in the IPv4/IPv6 pipeline 64 bytes of packet header is read if the packet length is greater than or equal to 60. Otherwise only 32 bytes are read.

Converting the packet flow shown in Figure 12-3 into a flattened Microblock Call graph in the dispatch loop may pose some challenges in that it may require more than 4K of control store (8K available in IXP2800 B0). [This will be done in an upcoming release] The current call graph for IPv4/MPLS and IPv4/IPv6 pipelines is:

- L2 Decap -> ILM -> IPv4 -> FTN -> Statistics
- L2 Decap -> IPv4 -> v6_decap -> IPv6 -> v6_encap -> IPv4 -> Statistics

Note: The IPv4 microblock is called twice in IPv4/IPv6 pipeline.

12.2.1.4 Statistics

This microblock runs on a single microengine. It is currently a placeholder for statistics handling. It is anticipated that when this application is extended for DiffServ, this microblock will be used to manage per-flow statistics. The design for handling statistics will be described in future releases of the document.

12.2.1.5 SPI4 TX

The SPI4 Transmit microblock transmits packets over SPI4 interface. It is implemented by extending the Packet TX microblock described in Section 5.2.2.5, “Packet TX” on page 72 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application” with a compile time option. It runs on two microengines in a context pipeline connected by a Next Neighbor ring. It segments a packet into m-packets, and moves them into TBUFS for the MSF state machine to transmit. The extensions to Packet TX are:

- TX to Scheduler feedback for flow control is disabled (using compile time option) as it’s not required in this case. (Use compile time option DISABLE_TX2SCHED_FEEDBACK)
- 16 bytes of per-packet header consisting of input and output port, color, class, flow id, nexthop id etc is pre-pended to start of the packet and is passed along to Ingress B. (Use compile time option SPI4_PREPEND)

12.2.2 Data Flow for the Ingress B IXP2800

12.2.2.1 SPI4 RX

The SPI4 RX microblock receives packets over the SPI4 interface. It is implemented by extending Packet RX microblock described in Section 5.2.1.1, “Packet RX” on page 67 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application” with a compile time option. It runs on two microengines in a context pipeline connected by a Next Neighbor ring. It performs frame reassembly on the m-packets coming in on POS media.

The extensions to Packet RX are:

- 16 bytes of per-packet header consisting of input and output port, color, class, flow id, nexthop id etc is received with every packet. This data is used to create meta-data for the packet and is written to SRAM after reassembly. Some of this info is passed along to the next microblock in the pipeline via a scratch ring. The rest of the packet is written to DRAM. (Use compile time option SPI4_PREPEND)

12.2.2.2 Meter & WRED

Diffserv capabilities will be available in an upcoming release. Until then a skeleton microblock with a complete dispatch loop is provided in the packet processing stage to pass the reassembled packet to statistics microblock.

12.2.2.3 Statistics Microblock

This block is identical to the Statistics block described in Section 5.2.1.3, “Statistics Microblock” on page 69 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application.”

12.2.2.4 CSIX Scheduler

This block is identical to the CSIX Scheduler block described in Section 5.2.1.4, “CSIX Scheduler” on page 69 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application.”

12.2.2.5 Cell Based Queue Manager (Cell QM)

This block is identical to the Cell Based Queue Manager block described in Section 5.2.1.5, “Cell Based Queue Manager (Cell QM)” on page 70 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application.”

12.2.2.6 CSIX TX

This block is identical to the CSIX TX block described in Section 5.2.1.6, “CSIX TX” on page 70 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application.”

12.2.3 Data Flow for the Egress IXP2800

This section describes the data flow for the Egress IXP2800. The egress pipeline is identical to the egress pipeline described in Section 5.2.2, “Data Flow for the Egress IXP2800” on page 71 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application.”

12.2.3.1 CSIX RX

This block is identical to the CSIX RX block described in Section 5.2.2.1, “CSIX RX” on page 71 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application.”

12.2.3.2 Statistics Microblock

This block runs on a single microengine. It is currently a place-holder for statistics handling. It is anticipated that when this application is extended for MPLS and DiffServ, this microblock will be used to manage per-flow statistics. The design for handling statistics will be described in future revisions of the document.

This microblock handles dropping of large packets that are stored in multiple buffers. It interfaces with Egress Packet Scheduler via the Next Neighbor ring.

12.2.3.3 Egress Packet Scheduler

The Egress scheduler schedules packets to be transmitted over MSF interface. This is a packet-based scheduler as opposed to the cell-based scheduler (i.e. c-frame) on the ingress side.

The packet scheduler is a context pipe-stage that is implemented as a microblock that runs on 3 microengines. This microblock includes the Class Scheduler block, the Count block, and the Port Scheduler block. Each block runs in one microengine.

The packet scheduler supports up to 16 virtual ports. Since these ports may have differing bandwidth requirements, the scheduler implements Weighted Round Robin (WRR) scheduling on the ports. This allows us to support different configurations (16 OC-3, 4 OC-12, 1 OC-48 etc) simply by adjusting the weights for the ports in the scheduler.

For each port, the scheduler supports up to 256 queues per port. The Scheduler implements a modified version of Deficit Round Robin (DRR) scheduling on the queues within a port. For more details, refer to Chapter 19, “OC-48 WRR/DRR Packet Scheduler” of *Intel® Internet Exchange Architecture Software Building Blocks Developer’s Manual*.

Until diffserv capabilities (Meter, WRED, 6-tuple classifier) are added the application will only use one class per port. This means there is only one queue per port and the DRR scheduling is unused. However the same code can be reused in a QoS Diffserv application in which case the DRR scheduling is applicable.

The scheduler also keeps track of the number of packets in flight (scheduled, but not transmitted) for each port. If this number exceeds a specified limit, then it stops scheduling on that port.

12.2.3.4 Packet Based Queue Manager (Packet QM)

This block is identical to the Packet QM described in Section 5.2.2.3, “Packet Based Queue Manager (Packet QM)” on page 72 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application.”

12.2.3.5 TX Helper

This block prepares TX request from the information passed from Packet QM via the Next Neighbor ring, and forwards the TX request to Packet TX through the scratch ring. It also updates the queue-based counters of packets in SRAM for Egress Packet Scheduler.

12.2.3.6 Packet TX

This is identical to the Packet TX as described in Section 5.2.2.5, “Packet TX” on page 72 in Chapter 5, “OC-192 POS IPv4/IPv6 Forwarding/Tunneling Application.”

12.3 Performance Characterization

The Intel® IXP2800 Network Processor operates at 1400 MHz. For a min POS packet of 49B, the packet inter-arrival time at OC-192 line rate is 57 ME cycles. In order to maintain line rate for min packets, each stage of the pipeline cannot exceed this budget. In other words, each stage of the pipeline needs to retire a packet every 57 cycles. Table 12-1 summarizes the performance analysis for the POS pipeline at OC192 rate.

Table 12-1. Performance Analysis for the POS Pipeline

OC-192c line rate assuming 3% SONET overhead	9.62 Gigabits/sec
Min POS packet size	49 bytes (40 byte TCP/IP, 2 bytes Address and Control, 2 byte PPP header, 4 byte FCS and 1 byte flag)
Packet Throughput for min packets	24.56 million packets/sec = (9.62 / (49*8)) * (10**9)
IXP2400 clock frequency	1400 MHZ
Inter-packet arrival time for min packets	1400/6.14 = 57 cycles
Compute cycles per packet for a single microengine	57
Latency per packet for a single microengine	57 * 8
Compute cycles per packet for n microengines running in parallel	57*n
Latency per packet for n microengines running in parallel	57*8*n

12.4 Ingress A System Resource Allocation

Table 12-2 shows how system resources are mapped for the Ingress A IXP2800 network processor. This mapping reflects the system defaults and may be changed. The allocation of microengines is done, such that it optimizes the performance of this specific application and may be changed for other applications.

Table 12-2. System Resources Mapped for the Ingress IXP2800

Microblock	ME #	Communication Mechanism with previous stage
Packet RX	ME 1:3, 1:4	Auto-push status from MSF, NN Ring
IPv4/IPv6/MPLS + Layer2 decapsulation/Classify	ME 0:0, 0:1, 0:2, 0:3, 0:4, 0:5, 0:6, 0:7	Scratch ring
Statistics	ME 1:0	Scratch ring
SPI 4 TX	ME 1:1, 1:2	NN Ring
Headroom	3 microengines	

The physical assignment of function to microengine is important since it not only affects when the next neighbor registers and signaling can be utilized, but it also affects the utilization of the internal Command Bus and S-Push/Pull buses. This assignment attempts to balance the usage of the Command bus and S-Push/Pull buses across the two clusters.

The IXP2800 supports four SRAM channels and three DRAM channels. Table 12-3 shows how the SRAM, DRAM and scratch are utilized for this application. These values are defined in a system header file `dl_system.h` and may be changed as needed.

Table 12-3. SRAM, DRAM, and Scratch Utilization for Ingress A IXP2800

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Buffer Descriptors	32	32k *(In simulation, we use only 320 buffers)*	1 MB		
Buffers	2048	32k		64 MB	
Trie Table	64 (The root Trie table requires at least 257k to support hi64k and hi256 tables. In addition each node requires 64 bytes. These nodes are added as needed)	See note in previous column. Assuming 256k routes, approximately 128k nodes are needed	8MB		
Route Table (Next Hop Information)	8	Assuming 4k next hops	32k		
IPv4 statistics	4	16			64
Packet RX statistics	4	16*16	1024		
IPv4 Directed Broadcast Table	32 (local memory)	64			
ILM NHLFE Table	24	64k	1.5 MB		
FTN NHLFE Table	24	64k	1.5MB		
Ring from Packet RX to packet processing pipeline (IPv4/IPv6/MPLS+Layer2 Decap/Classify)	12	2k/12			2k
IPv4 to Statistics ring	12	2k/12			2k
Buffer Free list Q-Array entry	N/A	4			

12.5 Ingress B System Resource Allocation

Table 12-4 shows how system resources are mapped for the Ingress B IXP2800.

Table 12-4. System Resources Allocated for Ingress B IXP2800

Microblock	ME #	Communication Mechanism with previous stage
SPI4 RX	ME 1:2, 1:3	Auto-push status from MSF, NN Ring
Meter/WRED (only skeleton for now)	ME 0:0, 0:1,	Scratch Ring
Statistics	MW 0:5	Scratch Ring

Table 12-4. System Resources Allocated for Ingress B IXP2800 (Continued)

Microblock	ME #	Communication Mechanism with previous stage
Fabric scheduler	ME 0:6	NN ring
QM	ME 0:7	NN Ring
CSIX TX	ME 1:0, 1:1	NN Ring
Headroom	7 microengines	

Table 12-5 shows how the SRAM, DRAM and scratch are utilized for this application. These values are defined in a system header file `dl_system.h` and may be changed as needed.

Table 12-5. SRAM, DRAM, and Scratch Utilization for Ingress B IXP2800

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Buffer Descriptors	32	32k *(In simulation, we use only 320 buffers)*	1 MB		
Buffers	2048	32k		64 MB	
Queue Descriptors	16	256 (1 per VOQ)	4K		
CSIX TX contexts	32	256 (1 per VOQ)	8k		
Ring from Packet RX to packet processing pipeline (IPv4+Layer2 Decap/ Classify)	12	2k/12			2k
QM Q-Array entries	N/A	16			
Buffer Free list Q-Array entry	N/A	4			

12.6 Egress System Resource Allocation

Table 12-6 shows how the system resources are allocated for the Egress IXP2800.

Table 12-6. System Resources Allocated for Egress IXP2800

Microblock	ME #	Communication Mechanism with previous stage
CSIX RX	ME 1:2, 1:3	Auto-push status from MSF, NN ring
Statistics	ME 0:2	Scratch ring
Egress Scheduler	ME 0:3, 0:4, 0:5	NN ring
Egress QM	ME 0:6	NN ring
TX Helper	ME 0:7	NN ring
Packet TX	ME 1:0, 1:1	NN ring
Headroom	6 microengines	N/A

The mapping of networking functions on to the microengines shows that 10 microengines are used to perform the fast path processing for this application. Additional functionality required by customers can be mapped on to the remaining microengines.

Table 12-7 shows how the SRAM, DRAM and scratch are utilized for this application. These values are defined in a system header file `dl_system.h` and may be changed as needed.

Table 12-7. SRAM, DRAM, and Scratch Utilization for Egress IXP2800

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM Used	Total Scratch used
Buffer Descriptors	32	32k *(In simulation we use only 320 buffers)*	1 MB		
Queue Descriptors	16	256 (16 ports x 16 classes per port)	4k		
CSIX RX Reassembly contexts	32	1024	32k		
Buffers	2048	32k		64 MB	
Inside Scheduler between Count block and Class Scheduler block	4	512			2k
TX Helper to first Packet TX	4	256			1K
TX Helper to second Packet TX	4	256			1K
QM Q-Array entries	N/A	16			
Buffer Free list Q-Array entry	N/A	4			

12.7 Interfaces Between the Various Microblocks

This section describes the interfaces between the different microblocks in the ingress and egress processors for this application.

12.7.1 Packet RX and Packet Processing Microengines

The interface between the Packet RX microblock and the packet processing microengines running the IPv4/IPv6/MPLS Forwarding and Layer-2 decap/classify microblocks is a scratch ring. Each entry in the scratch ring is three long words. The format depends on whether the packet fits in one buffer or not. In the case of packets that fit into a single buffer, all the information is packed into the scratch ring eliminating one read/write to SRAM in the critical path, see Table 12-8. In the case of packets that span across multiple buffers, some of the packet descriptor information is written to SRAM and the rest to the scratch ring, see Table 12-9. Bit 31 of LW0 (EOP bit of the handle) is used to detect if a packet spans across multiple buffers. If this bit is set (implying that the buffer is a SOP/EOP buffer), then the packet is contained in a single buffer.

This interface is used for packets that fit entirely in one buffer.

Table 12-8. Three-Word Scratch Ring Entry—Packets fit on one Buffer

LW	Bits	Size	Field	Description
0	31:0	32	dl_buffer_handle	Buffer Handle for the SOP Descriptor
1	31:16	16	input_port	Input port on ingress processor
	15:12	4	free_list_id	Free list ID for buffer
	11:8	4	rx_stat	Receive Status Flag
	7:0	8	header_type	Type of header at *offset* bytes into the packet
2	31:16	16	buffer_size	Buffer size in bytes
	15:0	16	offset	Offset of the start of data in the SOP buffer in bytes

This interface is used for packets that require more than one buffer.

Table 12-9. Three-Word Scratch Ring Entry—Packets fit on more than one Buffer

LW	Bits	Size	Field	Description
0	31:0	32	dl_buffer_handle	Buffer Handle for the SOP Descriptor
1	32:0	32	dl_eop_buffer_handle	Buffer Handle for the EOP Descriptor
2	31:16	16	packet_size	Total packet size across buffers in bytes
	15:0	16	offset	Offset of the start of data in the SOP buffer in bytes

12.7.2 Packet Processing Microengines and Statistics

The interface is a scratch ring. Each entry in the scratch ring is three long words as described in Table 12-10.

Table 12-10. Three-Word Scratch Ring Entry—Packet Processing Microengines and Statistics

LW	Bits	Size	Field	Description
0	30:16	16	MOP_EOP_buf_size	Size in bytes of all MOP buffers and the EOP buffer of the packet
0	0:15	16	Queue Number	Queue Number
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

12.7.3 Statistics and SPI4 TX

The interface is a NN ring. Each entry in the NN ring is three long words as described in Table 12-11.

Table 12-11. Three-Word NN Ring Entry (Statistics and SPI4 TX)

LW	Bits	Size	Field	Description
0	15:0	16	packet_len	Length of packet
0	31:16	16	Reserved	Not used

Table 12-11. Three-Word NN Ring Entry (Statistics and SPI4 TX)

1	31:0	32	Sop_handle	SOP Buffer handle
2	15:0	16	Port_number	Port Number
2	31:16	16	Queue Number	Queue Number

12.7.4 SPI4 RX and Meter/WRED

The interface is a scratch ring. Each entry in the scratch ring is three long words as described in Table 12-12.

Table 12-12. Three-Word Scratch Ring Entry (One Buffer only)

LW	Bits	Size	Field	Description
0	31:0	32	dl_buffer_handle	Buffer Handle for the SOP Descriptor
1	31:16	16	input_port	Input port on ingress processor
	15:12	4	free_list_id	Free list ID for buffer
	11:8	4	rx_stat	Receive Status Flag
	7:0	8	header_type	Type of header at *offset* bytes into the packet
2	31:16	16	buffer_size	Buffer size in bytes
	15:0	16	offset	Offset of the start of data in the SOP buffer in bytes

Table 12-13 shows the interface used for packets that require more than one buffer.

Table 12-13. Three-Word Scratch Ring Entry for SPI4 RX and Meter/WRED

LW	Bits	Size	Field	Description
0	31:0	32	dl_buffer_handle	Buffer Handle for the SOP Descriptor
1	32:0	32	dl_eop_buffer_handle	Buffer Handle for the EOP Descriptor
2	31:16	16	packet_size	Total packet size across buffers in bytes
	15:0	16	offset	Offset of the start of data in the SOP buffer in bytes

12.7.5 METER/WRED and Statistics

The interface is a scratch ring. Each entry in the scratch ring is three long words as described in Table 12-14.

Table 12-14. Three-Word Scratch Ring Entry for Meter/WRED and Statistics

LW	Bits	Size	Field	Description
0	30:16	16	MOP_EOP_buf_size	Size in bytes of all MOP buffers and the EOP buffer of the packet
0	0:15	16	Queue Number	Queue Number
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

12.7.6 Statistics and CSIX Scheduler

The interface is a next neighbor ring. Each entry in the NN ring is three long words as described in Table 12-15.

Table 12-15. Three-Word NN Ring Entry for Statistics and CSIX Scheduler

LW	Bits	Size	Field	Description
0	30:16	16	Packet cell count	Sum of all buffer cell counts belonging to the packet
0	0:15	16	Queue Number	Queue Number
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

12.7.7 CSIX Scheduler and Cell Queue Manager

The interface is a next neighbor ring. Each entry in the NN ring is three long words as described in Table 12-16.

Table 12-16. Three-Word NN Ring Entry for CSIX Scheduler and Cell Queue Manager

LW	Bits	Size	Field	Description
0	30:16	16	Dequeue Queue #	Queue number from which to dequeue. Zero implies no dequeue
0	0:15	16	Enqueue Queue #	Queue number on which to enqueue. Zero implies no enqueue
1	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
2	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

12.7.8 Cell Queue Manager and CSIX TX

The interface is a next neighbor ring. CSIX Transmit is a two-microengine context pipe-stage. The cell queue manager writes to the NN ring of the first CSIX TX microengine. Each entry in the NN ring is 2 words as described in Table 12-17.

Table 12-17. Two-Word NN Ring Entry for Cell Queue Manager and CSIX TX

LW	Bits	Size	Field	Description
0	31:16	16	Reserved	Reserved
0	15:0	16	Queue Number	Queue Number
1	31:0	32	Buffer Handle	Buffer Handle currently being transmitted for queue

12.7.9 CSIX TX—First ME to Second ME

The interface between the first CSIX TX microengine and second CSIX TX microengine is a next neighbor ring. Each entry in the NN ring is eight long words as described in Table 12-18.

Table 12-18. Eight-Word NN Ring Entry (CSIX TX—First ME to Second ME)

LW	Bits	Size	Field	Description
0	31:0	32	Tx_request0	Same as LW0 from Cell Queue Manager to CSIX TX
1	31:0	32	Tx_request1	Same as LW1 from Cell Queue Manager to CSIX TX
2	31:0	32	dram_handle	DRAM address where CSIX cell is stored
3	31:24	8	cell_count_remaining	Number of cells remaining in the current buffer
	23:18	6	Reserved	Reserved
	17:17	1	MOP_EOP_flag	If MOP_EOP, set to 1, else 0
	16:16	1	SOP_EOP_flag	If SOP and EOP, set to 0, else 1
	15:0	16	payload_length	Length of CSIX cell payload in bytes
4	31:0	32	prepend_header0	LW0 of CSIX cell pre-pend header
5	31:0	32	prepend_header1	LW1 of CSIX cell pre-pend header
6	31:0	32	prepend_header2	LW2 of CSIX cell pre-pend header
7	31:0	32	prepend_header3	LW3 of CSIX cell pre-pend header

12.7.10 CSIX RX and Statistics

The interface is a scratch ring. Each entry in the scratch ring is 3 words as described in Table 12-19.

Table 12-19. Three-Word Scratch Ring Entry for CSIX RX and Statistics

LW	Bits	Size	Field	Description
0	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
1	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)
2	30:16	15	Packet Size	Packet Size
2	15:12	4	Output Port Number	Output Port Number
2	11:0	12	Queue Number	Queue Number

12.7.11 Statistics and Packet Scheduler

The interface is a Next Neighbor ring as described in Table 12-20.

Table 12-20. Three-Word NN Ring Entry for Statistics and Packet Scheduler

LW	Bits	Size	Field	Description
0	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
1	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

Table 12-20. Three-Word NN Ring Entry for Statistics and Packet Scheduler (Continued)

LW	Bits	Size	Field	Description
2	30:16	15	Packet Size	Packet Size
2	15:12	4	Output Port Number	Output Port Number
2	11:0	12	Queue Number	Queue Number

12.7.12 Packet Scheduler and Queue Manager

The interface between the QM and the Packet Scheduler is a Next Neighbor Ring. Each entry is 2 long words as described in Table 12-21.

Table 12-21. Two-Word NN Ring Entry for Packet Scheduler and Queue Manager

LW	Bits	Size	Field	Description
0	30:16	16	Dequeue Queue #	Queue number from which to dequeue. Zero implies no dequeue
0	0:15	16	Enqueue Queue #	Queue number on which to enqueue. Zero implies no enqueue
1	31:0	32	SOP Buffer Handle	Buffer Handle for SOP Descriptor
1	31:0	32	EOP Buffer Handle	Buffer Handle for EOP Descriptor (may be NULL)

12.7.13 Queue Manager and TX Helper

The interface between the Queue Manager and the TX Helper a Next Neighbor ring. Each entry is two long words as described in Table 12-22.

Table 12-22. Two-Word NN Ring Entry for Queue Manager and TX Helper

LW	Bits	Size	Description
0	31:4	28	Reserved
0	3:0	4	Output port number
1	31:24	8	Reserved
1	23:0	24	Pointer to SOP buffer descriptor in SRAM in long words (Same as bits 0:23 of buffer handle)

12.7.14 TX Helper and Packet TX

The interface between the TX Helper and the Packet Transmit is two scratch rings – one for first Packet TX ME which handles the transmission of port 0 to 4, one for second Packet TX ME which handles the transmission of port 5 to 9. Each entry is one word as described in Table 12-23.

Table 12-23. One-Word Scratch Ring Entry for TX Helper and Packet TX

LW	Bits	Size	Description
0	31:31	1	Valid bit
	30:28	3	Reserved
	27:24	4	Port number
	23:0	24	Pointer to SOP buffer descriptor in SRAM in long words (Same as bits 0:23 of buffer handle)

Dual OC-12 POS/Dual Gb Ethernet Forwarding Application for IXDP24X1 13

This chapter describes an IPv4 Forwarding software application for Ethernet and Packet over SONET (POS) implemented on an Intel® IXP2400 Network Processor. It provides a high level design overview and lists the different software components used to build this application. This chapter describes the application in the context of Ethernet and POS media interfaces.

The application described in this chapter is supported on the Intel® IXDP2401 Advanced Development Platform, which uses a single IXP2400.

This chapter focuses only on the fast path or microengine components of the design. The Intel XScale® core components for this application are described in *Intel® Internet Exchange Architecture (IXA) Portability Framework Developer's Manual*.

Note: It is important that all applications developed for the IXDP24X1 platform must have the IX_PLATFORM_2401 flag defined in the project makefiles, for both the core components and the microblocks. An example of required flag definitions may be found in the makefiles of this application. By default, newly created projects under the Windriver* Tornado* development environment have the flag defined as IX_PLATFORM_2400. For this application, the flag must be changed to IX_PLATFORM_2401.

13.1 Hardware Overview

The Intel® IXDP2401 Advanced Development Platform consists of the Intel® IXMB2401 baseboard, which is equipped with two daughter board connectors (DB1 and DB2). Up to two media mezzanine boards can be connected to the baseboard. The following mezzanine boards are available:

- 2xOC-12 POS ATM mezzanine card
- 2x1 Gigabit Ethernet mezzanine card with copper interfaces
- 2x1 Gigabit Ethernet mezzanine card with fiber interfaces

The Intel® IXMB2401 baseboard may also be equipped with two types of front interfaces:

- 2x1 Gigabit Ethernet copper interfaces (MIC 2C)
- 2x1 Gigabit Ethernet fiber interfaces (MIC 2F)

Table 13-1 presents all possible hardware configurations supported by the Dual OC-12 POS/Dual Gigabit Ethernet Forwarding Application for IXDP2401.

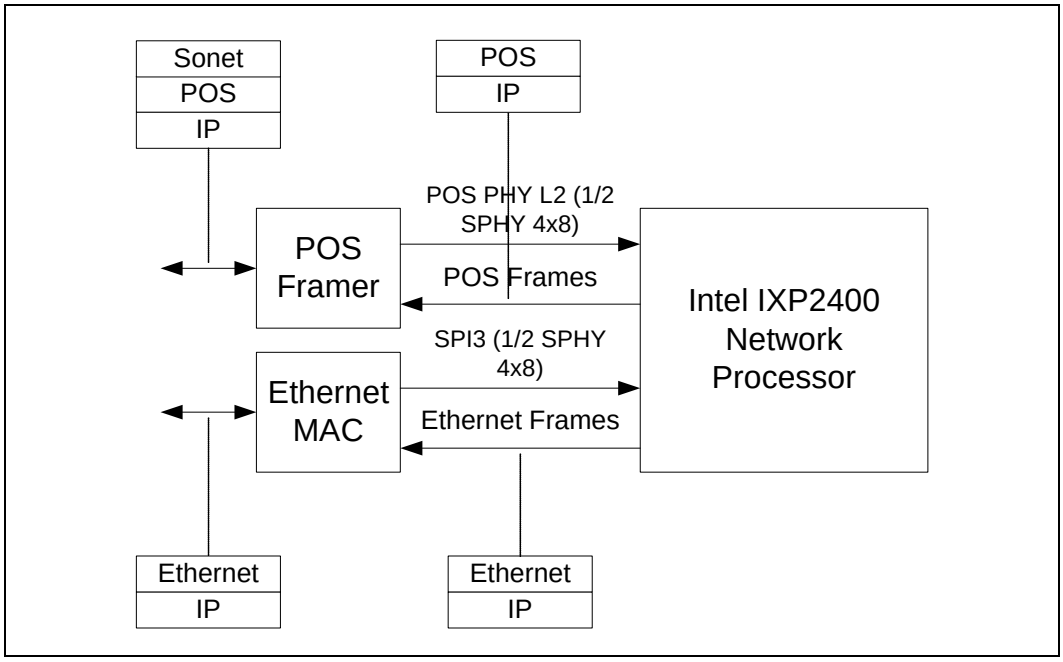
Table 13-1. Supported Hardware Configurations

Backplane (Interface Supported)	DB1 (Interface Supported)	DB2 (Interface Supported)	Description	Configuration with 100% Throughput
2x1GE (copper only)	2xOC-12 POS	N/A	All supported ports available.	1x1GbE, 2xOC-12 (due to 2.5 Gbps limitation for board)
N/A	MIC 2C 2x1GbE (copper only)	2xOC-12 POS	All supported ports available.	1x1GbE, 2xOC-12 (due to 2.5 Gbps limitation for board)
N/A	2xOC-12 POS	MIC 2F 2x1GbE (fiber only)	All supported ports available.	1x1GbE, 2xOC-12 (due to 2.5 Gbps limitation for board)
N/A	2xOC-12 POS	2x1GbE (mezzanine fiber or copper)	All supported ports available.	1x1GbE, 2xOC-12 (due to 2.5 Gbps limitation for board)

Figure 13-1 shows an Intel® IXP2400 Network Processor in a typical configuration. In this configuration, the IXP2400 is identified as the network processor; it receives from the Ethernet or POS media interface and transmits to the other Ethernet or POS media interface.

The target hardware comprises four physical media interfaces. A POS media mezzanine card installed on a baseboard provides two OC-12 interfaces. Two Gigabit Ethernet interfaces are available on the baseboard Backplane Access module.

Figure 13-1. Example Hardware Configuration for OC48-Ethernet/POS

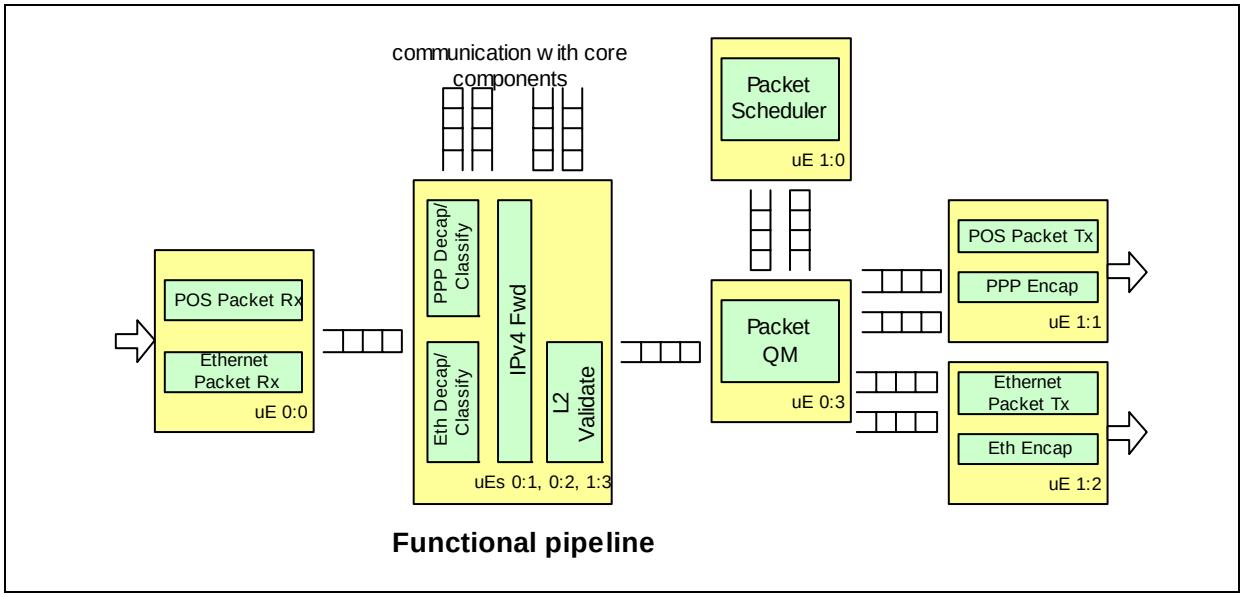


The IXP2400 receives POS or Ethernet frames that carry IPv4 datagrams. The frames are assembled into IPv4 packets and the Layer-2 (Ethernet or PPP) headers are removed. Based on the IPv4 header, a Longest Prefix Match (LPM) lookup is performed and the packets are transmitted over the appropriate port.

13.2 Software Overview

Figure 13-2 shows the microblocks needed to implement an OC-12 Ethenet/POS IPv4 Forwarding application. All the context pipe-stages (for example, Packet Rx, Queue Manager, and Scheduler) occupy an entire microengine. Each context pipe-stage is mapped to a single microblock running on a microengine with or without a dispatch loop. The functional pipeline runs on three microengines and implements decapsulation (Ethernet and PPP) together with decapsulation and the IPv4 forwarder blocks.

Figure 13-2. Microblocks for Dual OC-12 POS/ Dual Gigabit Ethernet IPv4 Forwarding Application



The design for the application shown in Figure 13-2 is based on the guidelines specified in the *Intel® Internet Exchange Architecture (IXA) Portability Framework Developer's Manual*. The driver microblocks (Receive, Transmit, Scheduler and Queue Manager) run on different microengines from the packet processing code. In this design, each driver block occupies an entire microengine. The packet processing blocks on the ingress IXP2400 include the IPv4 Forwarder and the PPP decapsulation/classify microblock. There are four microengines that run in parallel and execute the packet processing code. On the egress side, the only packet processing code is the PPP encapsulation block which runs on a single microengine.

13.2.1 Data Flow

This section describes the data flow on the Intel® IXP2400 Network Processor.

13.2.1.1 Ethernet Packet RX

The Ethernet Packet Receive (Rx) microblock performs frame-reassembly on the incoming mpackets on the media interface. It reassembles and writes the packet data to a buffer in DRAM and queues the packet buffer handle on a microengine-to-microengine scratch ring for processing

by the next stage of the pipeline. The Packet RX microblock also sets up per packet meta information (offset, size, etc.) which are passed down the pipeline either in a descriptor in SRAM or in the microengine-microengine scratch ring itself.

The Packet RX microblock runs on 4 threads on a single microengine (together with POS Rx). Each thread handles one micropacket (being in an RBUF) at a time. To maintain packet sequencing, the threads execute in strict order.

The Packet RX microblock works in the MSF mode. The SPI-3 bus is divided into four 8-bit SPI-3 connections. Since the Packet RX microblock uses 4 threads, each of 4 supported ports uses one thread for receiving packets. The re-assembly context for all these ports is kept in local memory.

From the Packet RX block, the packet is passed on to an application-specific system microblock. This microblock checks if the packet is marked to be dropped or sent to the Intel XScale® core. If not, it queues the packet buffer handle and associated meta-data into the scratch ring for the next stage in the pipeline.

13.2.1.2 POS RX

The POS Receive (Rx) is a driver microblock that performs frame-reassembly on the mpackets coming in on the POS media interface. It reassembles and writes the packet data to a buffer in DRAM and queues the packet buffer handle on a microengine-microengine scratch ring for processing by the packet processing microengine. The Packet RX microblock also sets up per packet meta information (offset, size, etc.) which are passed on either in a descriptor in SRAM or in the microengine-microengine scratch ring itself. In this application, the packets reassembled are PPP frames containing IP datagrams. RFC 2615 defines the Packet Over SONET specification and refers to RFC 1661 (PPP) and RFC 1662 (PPP in HDLC-like framing). PPP framing, including header validation, FCS generation and computation and byte stuffing, is handled by the POS framer (IXF 6048).

The Packet RX microblock uses 4 threads on a single microengine, each of which handles one mpacket at a time. In the application 2 ports are supported and the re-assembly context for all these ports is kept in local memory. To maintain packet sequencing, the threads execute in strict order.

Note: This microblock is written such that it supports up to 16 virtual ports, one or more of which may be unused. This allows the microblock to support different configurations such as Quad-OC12 or 16 OC-3 ports. However the application uses the block in the way that only 2 POS ports (OC-12 or OC-3) are supported.

Since POS packets may be up to 9k bytes, some large packets may be stored in multiple buffers chained together as a link-list. The buffer handles for the first and last packet in the chain are queued in the scratch ring.

From the Packet RX block, the packet is passed on to an application specific system microblock (`DL_Sink[]`). This microblock checks if the packet has been marked to be dropped (`IX_DROP`) or sent to the Intel XScale® core (`IX_EXCEPTION`). If not, it queues the packet buffer handle and associated packet meta data into the scratch ring for the next stage in the pipeline.

13.2.1.3 Ethernet Decapsulation and Classify

The Ethernet decapsulation/classify microblock runs in a functional pipeline with the PPP decapsulation, IPv4 and L2 validation microblocks on three microengines using 24 threads. This microblock classifies the incoming packets.

Before classification for all packets, their handlers are read from DRAM and cached in transfer registers. Packet descriptors are also cached in gprs.

Packet descriptor metadata is updated per decapsulation results. For an IP routed packet, the offset in buffer points at the IP header. IP ARP packets are passed to the Intel XScale® core as exception packets.

Subsequently, a packet is passed to the IPv4 forwarder, or to the Intel XScale® core (as an exception packet) for further processing.

13.2.1.4 PPP Decapsulation and Classify

The PPP decapsulation/classify microblock runs in a functional pipeline with the L2 decapsulation, IPv4 and L2 validation microblocks on three microengines using 24 threads.

An application specific system source microblock on each thread dequeues packet buffer handles from the scratch ring. This source block (`DL_Source[]`) is a system microblock implicit in the dispatch loop. It reads in the packet meta information—that is, the packet descriptor, and populates the dispatch loop state. It also reads in 32 bytes of the packet header from DRAM into a header cache maintained in transfer registers. Since it is important to maintain packet sequencing, the threads in the microblock execute in strict order to dequeue from the scratch ring. This implies that the first thread on microengine 1 dequeues the first packet, and signals the next thread to perform dequeue. From this block, the packet goes to the PPP decapsulation/classify microblock.

The PPP decapsulation/classify microblock removes the layer-2 PPP header from the packet by updating the offset and size fields in the packet meta descriptor. Based on the PPP header, it also classifies the packet into IPv4, PPP control packet (LCP, IPCP etc). If the packet is a PPP control packet, it is marked as an exception packet to be sent to the XScale Core (`IX_EXCEPTION`). Otherwise the packet is sent down the microengine pipeline for further processing. In this application, the dispatch loop will silently drop packets classified as IPv6.

13.2.1.5 IPv4 Forwarder

The IPv4 Forwarder microblock forwards IPv4 packets based on L3 addressing. The IPv4 Forwarder microblock uses a packet descriptor and accesses an IP header from the cache in the transfer registers. The IP packet is then validated against [RFC1812] and [RFC2644] within the data plane. If the IP packet fails any of the validation checks, the packet is dropped. The packet's IP header TTL is decremented, and the IP header checksum is updated accordingly. The packet's next hop is then determined (i.e., the next destination to which the packet is forwarded). To do that, the IP packet's destination address is passed to a 5-trie Longest Prefix Match (LPM) algorithm that yields a next hop index, which is used to obtain the next hop information. The information includes the output port and next hop ID, which is subsequently used to access the outgoing link layer information. The packet metadata is updated with the next hop ID, and the packet is handed off to the L2 Validation microblock. If the 5-trie algorithm fails (the best match cannot be determined), the packet is sent to the Intel XScale® core to complete the LPM procedure.

13.2.1.6 Packet-Based Queue Manager

The Packet-Based Queue Manager (QM) performs enqueue/dequeue operations on the hardware-assisted SRAM queues for packet-type traffic. The QM receives enqueue requests from the IPv4 microblock through a scratch ring. When the queue state changes between empty and non-empty, QM sends a transition message to the scheduler (via next neighbor registers). After every dequeue operation, the QM passes a transmit request to the scratch ring served by the Packet TX microblock. Dequeue requests come from the packet scheduler microengine.

13.2.1.7 Packet Scheduler

The Packet Scheduler selects constant-length packet segments to be transmitted to the MSF interface. The Packet Scheduler sends the Queue Manager microblock a message to dequeue a packet from a specific port's queue. The Queue Manager microblock services the request, and deposits a packet descriptor from the requested queue into the output packet ring.

The scheduler employs Round Robin (RR) algorithm among the output ports and Weighted Round Robin (WRR) algorithm among the port queues. Using the Weighted Round Robin algorithm on the 16 virtual ports allows us the flexibility to support a number of different configurations such 16 OC-3, 3 OC-12, and 4 OC-3, etc. The weights on the ports are adjusted according to the data rate sustained on that port.

To prevent head-of-line blocking, the scheduler with the help of feedback from the Packet TX block keeps track of the number of packets in flight (scheduled, but not transmitted) for each port. If this number exceeds a specified limit, then it stops scheduling on that port.

13.2.1.8 Ethernet Encapsulation

Ethernet encapsulation conditionally adds an appropriate layer-2 Ethernet header to the packet payload while copying it to a set of TBUFs. If the next hop id is set to an invalid value (-1), the block assumes that the layer-2 header has already been added to the packet and simply the packet is copied to a set of TBUFs without changes.

Ethernet encapsulation is integrated within the Ethernet Packet TX microblock.

13.2.1.9 Ethernet Packet TX

The Ethernet Packet TX microblock transmits Ethernet frames via the MSF interface as one or more consecutive mpackets (containing elements/segments of Ethernet frames). The Ethernet TX microblock fetches a packet buffer handle (to access an upstream packet descriptor) from the per port assigned packet ring (i.e., scratch memory ring); the packet descriptor references the payload of an Ethernet frame. Using the supplied context, the Ethernet TX microblock proceeds to transmit frame mpackets out the output port. Upon transmitting all MPKT frames, the packet buffer(s) is recycled.

The Ethernet Packet TX is used in the way that it supports up to 2 Ethernet ports. The transmit context for all of these are kept in local memory. Therefore the CAM is not required. The microblock monitors the MSF to see if the TBUF threshold for a specific port has been exceeded. If so, it stops transmitting on that port and any requests to transmit packets on that port are queued up in local memory.

The Packet TX microblock periodically updates the scheduler with information about how many packets have been transmitted. If the packets in flight for a particular port (packets scheduled but not transmitted) exceed a certain limit (which depends on the bandwidth supported by that port), then the scheduler stops scheduling any more packets for the port. This combination of queuing packets in local memory and keeping track of the packets in flight helps prevent head-of-line blocking.

The Packet TX microblock runs on two microengines and supports SPHY 4x8 configuration.

13.2.1.10 PPP Encapsulation

This block conditionally adds the layer-2 PPP header to the packet while copying it to a set of TBUFs. If the next hop id in the packet meta data is set to an invalid value (-1) then the block assumes that the PPP header has already been added to the packet and is simply copied to a set of TBUFs without changes.

PPP encapsulation is integrated within the POS Packet TX microblock.

13.2.1.11 POS Packet TX

The POS Packet TX microblock transmits packets over the media interface. It segments a packet into mpackets and moves them into TBUFS for the MSF state machine to transmit. The POS Packet TX microblock assumes that the layer-2 header is already prepended to the start of the packet by a previous stage of the packet processing pipeline. It also receives a transmit request for the entire packet.

The POS Packet TX microblock is set up to support up to 2 POS ports. The transmit context for all of these are kept in local memory. Therefore the CAM is not required. The microblock monitors the MSF to see if the TBUF threshold for a specific port has been exceeded. If so it stops transmitting on that port and any requests to transmit packets on that port are queued up in local memory.

The POS Packet TX microblock periodically updates the scheduler with information about how many packets have been transmitted. If the packets in flight for a particular port (packets scheduled but not transmitted) exceed a certain limit (which depends on the bandwidth supported by that port), then the scheduler stops scheduling any more packets for the port. This combination of queuing packets in local memory and keeping track of the packets in flight helps prevent 'head of line blocking'.

An assumption made in this design is that the output port for egress is found via the IPv4 lookup performed on the ingress side. A different approach is to use the next hop id and do a lookup on the egress side to find out the output port number.

The POS Packet TX microblock runs on a single microengine together with PPP encapsulation.

Note: The POS Packet TX microblock can be used to support the MPHY-4 (or SPHY 4x8—four port OC-12) configuration when it runs on two microengines. However, in this application it runs on a single microengine in SPHY 4x8 mode so that only 2 ports (OC-12 or OC-3) are supported.

13.2.2 Dispatch Loops

There are four microblock groups, called dispatch loops, used in this pipeline application. For more information on dispatch loops, refer to the *Intel® Internet Exchange Architecture (IXA) Portability Framework Developer's Manual* "Dispatch Loop" chapter.

- Dispatch Loop for the Packet Frame Reassembly Stage (Figure 13-3)
- Dispatch Loop for the IPv4 Forwarder functional pipeline (Figure 13-4)
- Dispatch Loop for the PPP transmit stage (Figure 13-5)
- Dispatch Loop for the Ethernet transmit stage (Figure 13-6)

The QM and Scheduler blocks do not use a dispatch loop, though they still use the dispatch loop macros where required

Note: The system microblocks `dl_source, dl_sink, dl_qm_sink`, etc are application-specific. They may be changed for different packet processing pipelines.

Figure 13-3. Dispatch Loop for the Packet Frame Reassembly Stage

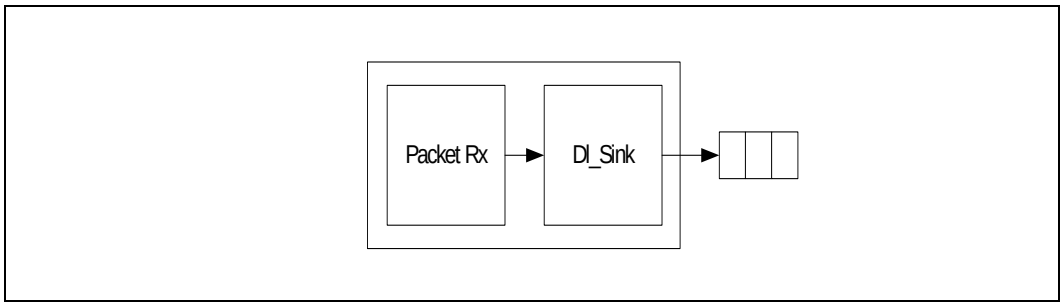


Figure 13-4. Dispatch Loop for the IPv4 Functional Pipeline

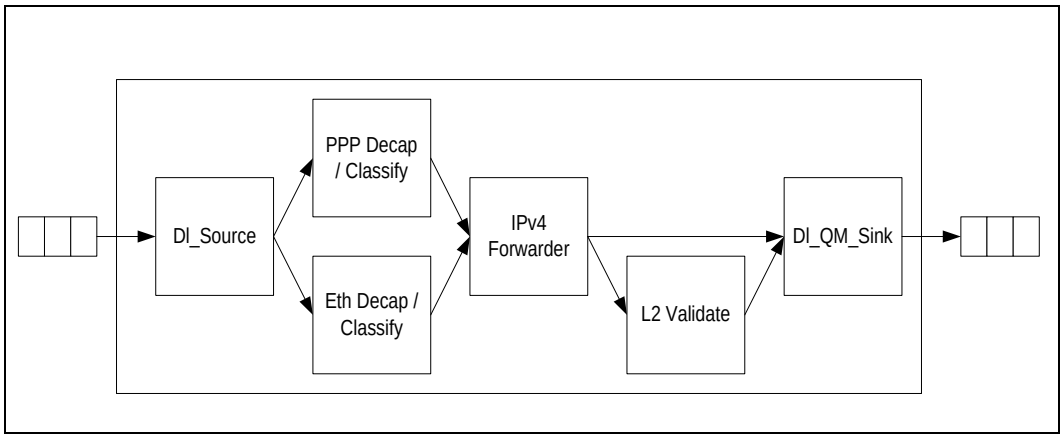


Figure 13-5. Dispatch Loop for POS Transmit Stage

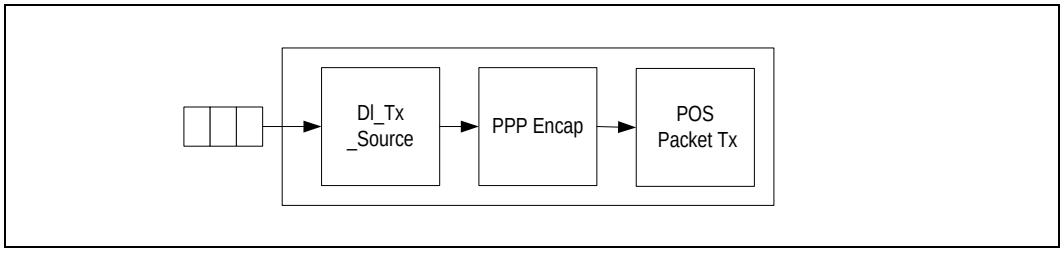
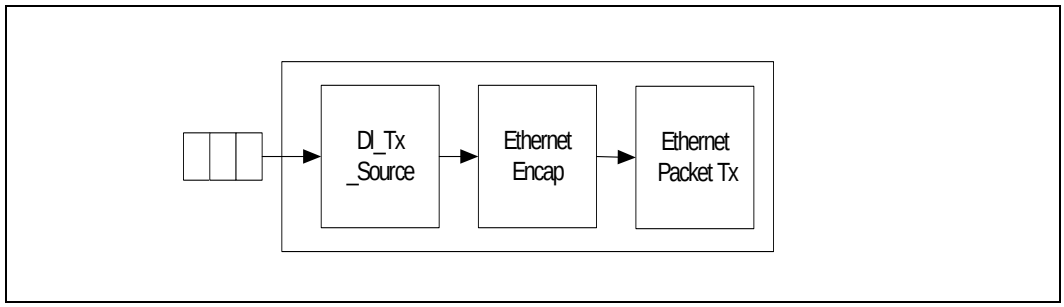


Figure 13-6. Dispatch Loop (Microblock group) for Ethernet Transmit Stage



13.3 Performance Characterization

13.3.1 POS/Ethernet Pipeline

The Intel® IXP2400 Network Processor operates at 600 MHz. The application handles two OC-12 POS ports and two Gigabit Ethernet ports. For a minimum POS packet of 49B, the packet inter-arrival time at two OC-12 line rate is 194 microengine cycles. For a minimum Ethernet packet of 64B with extra gap of 20B (looks like 84 bytes on a wire) the packet inter-arrival time at two Gigabit Ethernet port is 200 microengine cycles. In order to maintain line rate for minimum packets on all four ports, each stage of the pipeline cannot exceed the budget following average value = (195,4 + 200)/4 = 98,85 microengine cycles . In other words, each stage of the pipeline needs to retire a packet every 98 cycles.

Table 13-2 summarizes the performance analysis for the POS pipeline.

Table 13-2. Performance Characterization for the POS Pipeline

Parameter	Value
Summarized dual OC-12 line rate with dual Gigabit Ethernet line rate	3.204 Gigabits/sec
Minimum POS packet size	49 bytes (40 byte TCP/IP, 2 bytes Address and Control, 2 byte PPP header, 4 byte FCS and 1 byte flag)
Packet throughput for min packets	6.05 million packets/sec = (1.204 / (49*8)) * (10**9) + (2 * 1.000 /(84*8) * (10**9)
IXP2400 clock frequency	600 MHZ
Inter-packet arrival time for min packets	600/6.05 = 98.8 cycles
Compute cycles per packet for a single microengine	98
Latency per packet for a single microengine	98 * 8
Compute cycles per packet for n microengines running in parallel	98 * n
Latency per packet for a n microengines running in parallel	98 * 8 * n

13.4 System Resource Allocation

Table 13-3 shows the system resources mapped for the Intel® IXP2400 Network Processor. This mapping reflects the system defaults and may be changed. The allocation of microengines is done such that it optimizes the performance of this specific application and may be changed for other applications.

The physical assignment of function to microengine is important since it not only affects when the next neighbor registers and signaling can be utilized, but it also affects the utilization of the internal command bus and S-Push/Pull buses. Since ME0-ME3 belong to Microengine Cluster 0 and ME4-ME7 belong to Microengine Cluster 1, this assignment attempts to balance the usage of the command bus and S-Push/Pull buses across the two clusters.

The IXP2400 supports two SRAM channels and one DRAM channel. Table 13-4 shows the SRAM, DRAM, and scratch memory utilized for this application. These values are allocated by the Intel XScale® core application and patched to the microcode upon startup. The values are defined in a system XML configuration file `ix_sa_registry.xml` and may be changed as required.

Table 13-3. System Resources Mapped for the IXP2400

Microblock	ME #	Communication
Packet Rx	ME0	Auto-push status from MSF
IPv4 Forwarder + Eth Decapsulation/ Classifier + PPP Decapsulation/ Classifier + L2 Validation	ME1, ME2, M7	Scratch Ring
Queue Manager	ME3	Scratch Ring
Packet Scheduler	ME4	Scratch Ring
PPP Packet TX + PPP Encapsulation	ME5	Scratch Ring
Ethernet Packet TX + Eth Encapsulation	ME6	Scratch Ring

Table 13-4. SRAM, DRAM, and Scratch Utilization for Ingress Resource Allocation

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Buffer Descriptors	32	16000	512000	-	-
Buffers	2048	16000	-	32768000	-
Queue Descriptors	16	1025	16400	-	-
Layer-2 table with mapping from next hop id to Ethernet header	16	1024	16384	-	-
Trie Table	64B for a trie entry	1024 used for approximation	256kB+1KB+1/ 2*1024*64B= 295936	-	-
Route Table (Next Hop Information)	16	1024	-	16384	-
IPv4 statistics	32	16 (needed 4)	-	-	512

Table 13-4. SRAM, DRAM, and Scratch Utilization for Ingress Resource Allocation (Continued)

IPv4 Directed Broadcast Table	32	256 + 32 extra	9216	-	-
Processing Control Block	4	1	4	-	-
Ring from Packet RX to packet processing pipeline (IPv4+Eth Decap/ Classify)	5*4B=20	204	-	-	1024*4B= 4096
IPv4 to QM ring	3*4B=12	170	-	-	512*4= 2048
Scheduler to QM ring	1*4B=4	512	-	-	512*4= 2048
QM Q-Array entries	4	16	64	-	-
Buffer Free list Q-Array entry	4	4	16	-	-
uCode to Xscale Core priority rings (1 AF+ 1 BE)	2*4B	64	-	-	2*512 = 1024
Xscale Core to uCode rings (1 CNTRL + 1 DATA)	1*4B, 2*4B	128	-	-	128*4+ 256*4= 1536
QM to Packet TX rings	1*4B=4	256	-	-	4*256*4B= 4096

13.5 Microblock Interface

This section describes the interfaces between the different microblocks for this pipeline application.

In most of the messages, there is a valid bit is used to prevent a value of zero from being enqueued on the scratch ring. Zero is used to detect a case where the scratch ring is empty. So the valid bit helps distinguish between a zero value that was actually enqueued versus a case where the ring is empty.

13.5.1 Packet RX and Packet Processing Microengines

The interface between the Packet Receive microblock and the Packet Processing microengines (IPv4 Forwarder + L2/PPP decap + L2 Validate) is a scratch ring. Table 13-5 describes each entry in the scratch ring—which is five words.

Table 13-5. Packet RX and Packet Processing Microengines Scratch Ring Interface

LW	Bits	Size	Field	Description
0	31:0	32	dl_buffer_handle	Buffer handle for the SOP descriptor
1	31:0	32	dl_eop_buffer_handle	Buffer handle for the EOP descriptor
2	31:!6	16	buffer_size	Buffer size in bytes
	15:0	16	offset	Offset of the start of data in the buffer in bytes
3	31:16	16	packet_size	Total packet size across buffers
	15:12	4	free_list_id	Free list ID for buffer

Table 13-5. Packet RX and Packet Processing Microengines Scratch Ring Interface

	11:8	4	rx_stat	Receive Status Flag
	7:0	8	header_type	Type of header at offset bytes into the packet
4	31:16	16	input_port	Input port on the Network Processor
	15:0	16	reserved	Reserved

13.5.2 Packet Processing Microengines and Packet QM

The interface between the Packet Processing microengines (IPv4 Forwarder + L2/PPP decap + L2 Validate) and Packet QM is a scratch ring. Table 13-6 describes each entry in the scratch ring—which is five words.

Table 13-6. Packet Processing Microengines and Packet QM Scratch Ring Interface

LW	Bits	Size	Field	Description
0	31:0	32	SOP Buffer Handle	Buffer Handle for the SOP Descriptor
1	31:0	32	EOP Buffer Handle	Buffer Handle for the EOP Descriptor (can be NULL)
2	31	1	Valid Bit	Must be 1
2	30:16	15	Reserved	Reserved
2	15:0	16	Queue Number	Queue Number

13.5.3 Packet Queue Manager and Scheduler

The interface between the packet-based Queue Manager and the POS/Ethernet Scheduler is a Next Neighbor Ring.

Table 13-7. Queue Transition Messages Sent by the Packet Queue Manager

LW	Bits	Size	Field	Description
0	31	1	Valid Bit	Must be 1
	30	1	Enqueue Transition	Notification that queue has gone from empty to non-empty
	29	1	Reserved	Reserved
	28:18	11	Packet Cell Count	Unused for POS/Ethernet
	17:16	2	Reserved	Reserved
	15:0	16	Queue Number	Queue Number that was enqueued (only 8 bits used for POS/Ethernet)
1	31	1	Valid Bit	Must be 1
	30	1	Dequeue Transition	Notification that queue has gone from non-empty to empty
	29	1	Invalid Dequeue	If set, then dequeue request to an invalid queue was made
	28:16	13	Packet Size	Packet size in 128 bytes units (only 7 bits used)
	15:0	16	Queue Number	Queue number that was dequeued (only 8 bits used for POS/Ethernet)

13.5.4 Packet Queue Manager and POS Packet TX

The interface between the Packet Queue Manager and the POS Packet Transmit microengines is two scratch rings—one per OC-12 port. Table 13-8 describes each entry in the scratch ring—which is one word.

Table 13-8. Packet Queue Manager and Packet TxScratch Ring Interface

LW	Bits	Size	Field	Description
0	31	1	Valid Bit	Must be 1
	30:28	3	Reserved	Reserved
	27:24	4	Port Number	Port Number
	23:0	24	SOP Buffer Descriptor	Pointer to SOP buffer descriptor in SRAM in long words (same as bits 23:0 of buffer handle)

13.5.5 Packet Queue Manager and Ethernet Packet TX

The interface between the Packet Queue Manager and the Ethernet Packet Transmit microengines is two scratch rings—one per Gigabit Ethernet port. Table 13-9 describes each entry in the scratch ring—which is one word.

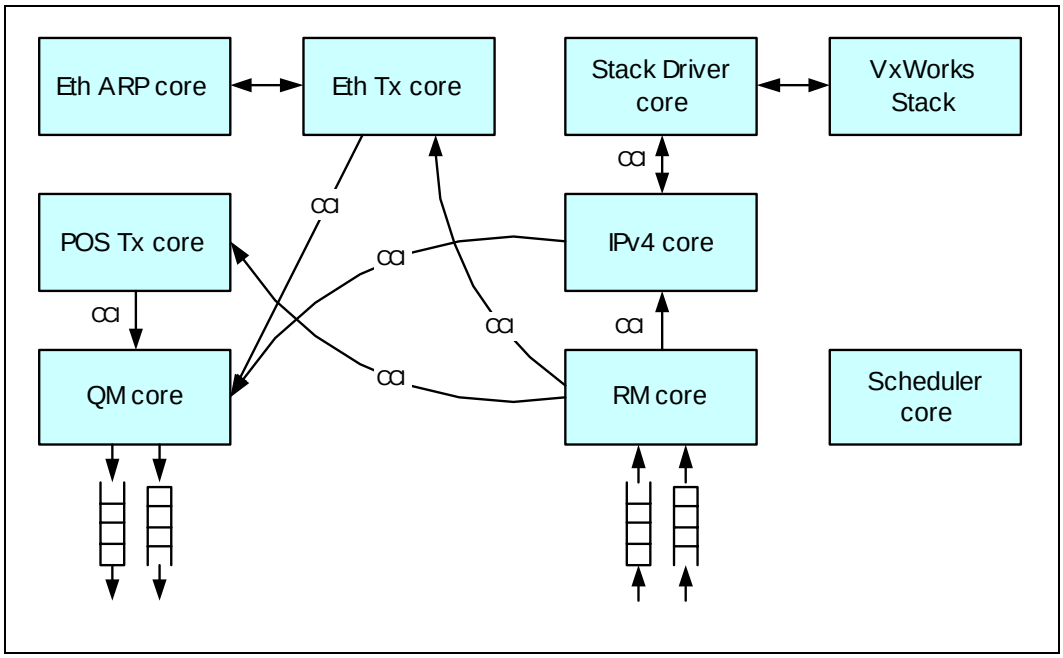
Table 13-9. One-word Scratch Ring (Packet Queue Manager and Packet TX)

LW	Bits	Size	Field	Description
0	31	1	Valid Bit	Must be 1
	30:28	3	Reserved	Reserved
	27:24	4	Port Number	Port Number
	23:0	24	SOP Buffer Descriptor	Pointer to SOP buffer descriptor in SRAM in long words (same as bits 23:0 of buffer handle)

13.6 Core Components Usage

The Dual OC-12 POS/ Dual Gigabit Ethernet Forwarding pipeline application uses standard core components customized to use only channel 0 for SRAM. Figure 13-7 shows the interconnections between the application's core components. The Resource Manager and Queue Manager core components employ scratch rings for communication with microblocks on microengines. Interactions between IPv4, Ethernet Tx, POS Tx, Stack Driver, Resource Manager and Queue Manager are managed by the Core Component Interface (CCI).

Figure 13-7. Core Components in the OC-12 POS/Ethernet IPv4 Forwarding Application



Quad Gigabit Ethernet Forwarding Application for IXDP24X1 14

This chapter describes a Quad Gigabit Ethernet Forwarding pipeline application implemented on one Intel® IXP2400 Network Processor. This chapter contains a high-level design overview and lists the different software components used to build this pipeline application.

The application described in this chapter is supported on the Intel® IXDP2401 Advanced Development Platform, which uses a single IXP2400.

This chapter focuses only on the fast path or microengine components of the design. The Intel XScale® core components for this application are described in the *Intel® Internet Exchange Architecture (IXA) Portability Framework Developer's Manual*.

Note: It is important that all applications developed for the IXDP24X1 platform must have the IX_PLATFORM_2401 flag defined in the project makefiles, for both the core components and the microblocks. An example of required flag definitions may be found in the makefiles of this application. By default, newly created projects under the Windriver* Tornado* development environment have the flag defined as IX_PLATFORM_2400. For this application, the flag must be changed to IX_PLATFORM_2401.

14.1 Hardware Overview

The Intel® IXDP2401 Advanced Development Platform consists of the Intel® IXMB2401 baseboard, which is equipped with two daughter board connectors (DB1 and DB2). Up to two media mezzanine boards can be connected to the baseboard. The following mezzanine boards are available:

- 2x1 Gigabit Ethernet mezzanine card with copper interfaces
- 2x1 Gigabit Ethernet mezzanine card with fiber interfaces

The Intel® IXMB2401 baseboard may also be equipped with two types of front interfaces:

- 2x1 Gigabit Ethernet copper interfaces (MIC 2C)
- 2x1 Gigabit Ethernet fiber interfaces (MIC 2F)

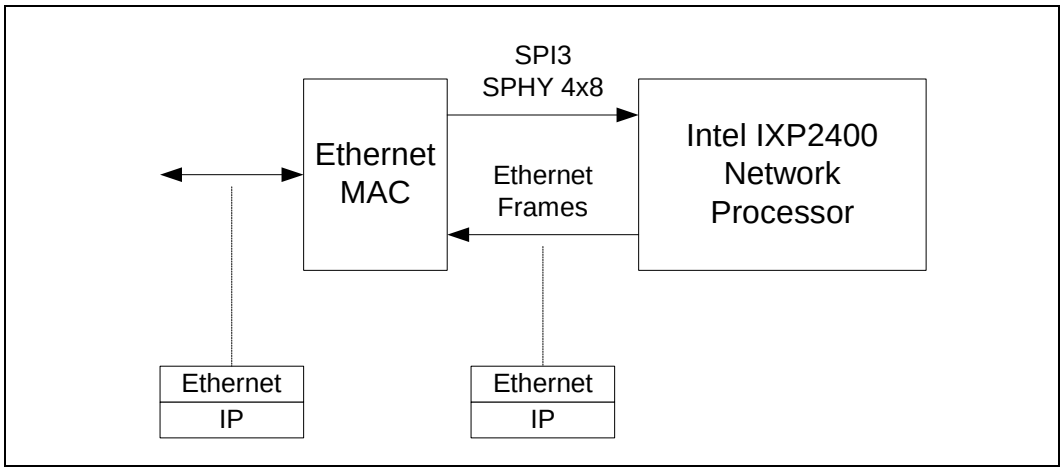
Table 14-1 presents all possible hardware configurations supported by the Quad Gigabit Ethernet Forwarding Application for IXDP2401.

Table 14-1. Supported Hardware Configurations

Backplane (Interface Supported)	DB1 (Interface Supported)	DB2 (Interface Supported)	Description	Configuration with 100% Throughput
2x1GE (copper only)	MIC 2C 2x1GbE (copper only)	N/A	All supported ports available.	1x1GbE front, 1x1GbE backplane (due to 2.5 Gbps limitation for board)
2x1GE (copper only)	N/A	MIC 2F 2x1GbE (fiber only)	All supported ports available.	1x1GbE front, 1x1GbE backplane (due to 2.5 Gbps limitation for board)
2x1GE (copper only)	2x1GbE (mezzanine fiber or copper)	N/A	All supported ports available.	1x1GbE front, 1x1GbE backplane (due to 2.5 Gbps limitation for board)

The application runs on an IXMB2401 baseboard with an Intel® IXP2400 Network Processor. The baseboard is equipped with two front panel Gigabit Ethernet interfaces (copper or fiber) as well as two backplane Gigabit Ethernet interfaces. Additionally, a serial console port and debug Ethernet port are available. The baseboard is designed according to ATCA standards and requires an ATCA-compliant chassis.

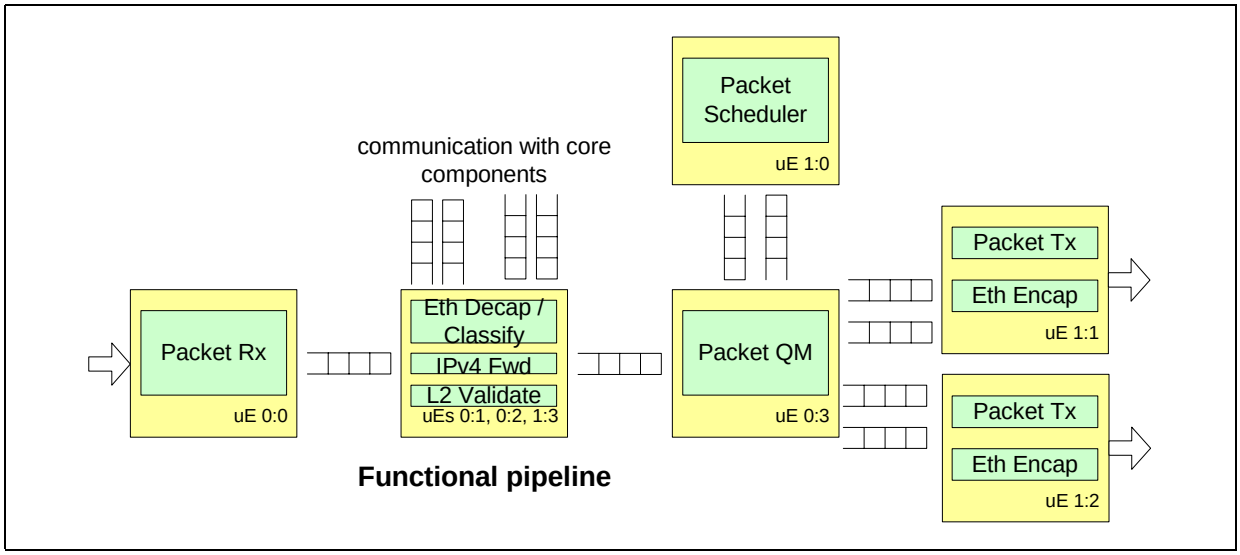
Figure 14-1. Example Hardware Configuration for Quad Ethernet IPv4 Forwarding Application



14.2 Software Overview

Figure 14-2 shows the microblocks needed to implement a Quad Ethernet IPv4 Forwarding pipeline application. All the context pipe-stages (for example, Packet Rx, Queue Manager, and Scheduler) occupy an entire microengine. Each context pipe-stage is mapped to a single microblock running on a microengine with or without a dispatch loop. The functional pipeline runs on three microengines and implements the layer-2 (Ethernet) decapsulation and the IPv4 forwarder blocks.

Figure 14-2. Microblocks for a Quad Ethernet IPv4 Forwarding Application



14.2.1 Data Flow

14.2.1.1 Packet RX

The Packet Receive (RX) microblock performs frame-reassembly on the incoming packets on the media interface. It reassembles and writes the packet data to a buffer in DRAM and queues the packet buffer handle on a microengine-microengine scratch ring for processing by the next stage of the pipeline. The Packet RX microblock also sets up per packet meta information (offset, size, etc.) which are passed down the pipeline either in a descriptor in SRAM or in the microengine-microengine scratch ring itself.

The Packet RX microblock runs on 8 threads on a single microengine. Each thread handles one micropacket (rbuf) at a time. To maintain packet sequencing, the threads execute in strict order.

The Packet RX microblock works in MSF mode. The SPI-3 bus is divided into four 8-bit SPI-3 connections. Since the Packet RX microblock uses 8 threads, each of 4 supported ports uses two threads for receiving packets. The re-assembly context for all these ports is kept in local memory.

From the Packet RX block, the packet is passed on to an application-specific system microblock. This microblock checks if the packet is marked to be dropped or sent to the Intel XScale® core. If not, it queues the packet buffer handle and associated meta-data into the scratch ring for the next stage in the pipeline.

14.2.1.2 Ethernet Classify/Decapsulate

The Ethernet decapsulation/classify microblock runs in a functional pipeline with the IPv4 microblock on three microengines using 23 threads. This microblock classifies the incoming packets.

Before classification for all packets, their handlers are read from DRAM and cached in transfer registers. Packet descriptors are also cached in gprs.

Packet descriptor metadata is updated per decapsulation results. For an IP routed packet, the offset in buffer points at the IP header. IP ARP packets are passed to the Intel XScale® core as exception packets. Subsequently, a packet is passed to the IPv4 forwarder, or to the XScale core (as an exception packet) for further processing.

14.2.1.3 IPv4 Forwarder

The IPv4 Forwarder microblock forwards IPv4 packets based on L3 addressing. The IPv4 Forwarder microblock uses a packet descriptor and accesses an IP header from the cache in the transfer registers. The IP packet is then validated against [RFC1812] and [RFC2644] within the data plane. If the IP packet fails any of the validation checks, the packet is dropped. The packet's IP header TTL is decremented, and the IP header checksum is updated accordingly. The packet's next hop is then determined (that is, the next destination to which the packet is forwarded). To do that, the IP packet's destination address is passed to a 5-trie Longest Prefix Match (LPM) algorithm that yields a next hop index, which is used to obtain the next hop information. The information includes the output port and next hop ID, which is subsequently used to access the outgoing link layer information. The packet metadata is updated with the next hop ID, and the packet is handed off to the L2 Validation microblock. If the 5-trie algorithm fails (the best match cannot be determined), the packet is sent to the Intel XScale® core to complete the LPM procedure.

14.2.1.4 L2 Validate

The L2 Validate microblock checks for outgoing packets if layer-2 Ethernet header exists in the L2 Table. If the header is not found, the packet is enqueued to be processed by the XScale Core. ARP Processing is handled by the XScale application code.

The L2 Validate microblock is located after IPv4 Forwarder in the functional pipeline.

14.2.1.5 Packet-Based Queue Manager

The Packet-Based Queue Manager (QM) performs enqueue/dequeue operations on the hardware-assisted SRAM queues for packet-type traffic. The QM receives enqueue requests from the IPv4 microblock through a scratch ring. When the queue state changes between empty and non-empty, QM sends a transition message to the scheduler (via next neighbor registers). After every dequeue operation, the QM passes a transmit request to the scratch ring served by Packet TX microblock. Dequeue requests come from the packet scheduler microengine.

14.2.1.6 Packet Scheduler

The Packet Scheduler selects constant-length packet segments to be transmitted to the MSF interface. The Packet Scheduler sends the Queue Manager microblock a message to dequeue a packet from a specific port's queue. The Queue Manager microblock services the request, and deposits a packet descriptor from the requested queue into the output packet ring.

The scheduler employs Round Robin (RR) algorithm among the output ports and Weighted Round Robin (WRR) algorithm among the port queues.

To prevent head-of-line blocking, the scheduler with the help of feedback from the Packet TX block keeps track of the number of packets in flight (scheduled, but not transmitted) for each port. If this number exceeds a specified limit, then it stops scheduling on that port.

14.2.1.7 Ethernet Encapsulation

Ethernet encapsulation conditionally adds an appropriate layer-2 Ethernet header to the packet payload while copying it to a set of TBUFs. If the next hop id is set to an invalid value (-1), the block assumes that the layer-2 header has already been added to the packet and the packet is simply copied to a set of TBUFs without changes.

Ethernet encapsulation is integrated within the Ethernet Packet TX microblock and runs on the same two microengines.

14.2.1.8 Packet TX

The Packet TX microblock transmits Ethernet frames via the MSF interface as one or more consecutive MPKTs (containing elements/segments of Ethernet frames). The Ethernet TX microblock fetches a packet buffer handle (to access an upstream packet descriptor) from the egress packet ring (i.e., scratch memory ring); the packet descriptor references the payload of an Ethernet frame. Using the supplied context, the Ethernet TX microblock proceeds to transmit frame MPKTs out the egress port. Upon transmitting all MPKT frames, the packet buffer(s) is recycled.

The Packet TX microblock periodically updates the scheduler with information about how many packets have been transmitted. If the packets in flight for a particular port (packets scheduled but not transmitted) exceed a certain limit (which depends on the bandwidth supported by that port), then the scheduler stops scheduling any more packets for the port. This combination of queuing packets in local memory and keeping track of the packets in flight helps prevent head-of-line blocking.

The Packet TX microblock runs on two microengines and supports SPHY 4x8 configuration. Thus 4 Gigabit Ethernet ports are supported.

14.2.2 Dispatch Loops

There are three dispatch loops on the application's pipeline:

- Dispatch Loop for the Packet Frame Reassembly Stage (Figure 14-3)
- Dispatch Loop for the IPv4 Forwarder functional pipeline (Figure 14-4)
- Dispatch Loop for the Ethernet transmit stage (Figure 14-5)

The QM and Scheduler blocks do not use a dispatch loop (they still use the dispatch loop macros where required).

Note: Note that the system microblocks `dl_source, dl_sink, dl_qm_sink`, etc. are application-specific. They may be changed for different packet processing pipelines.

Figure 14-3. Dispatch Loop for the Packet Frame Reassembly Stage

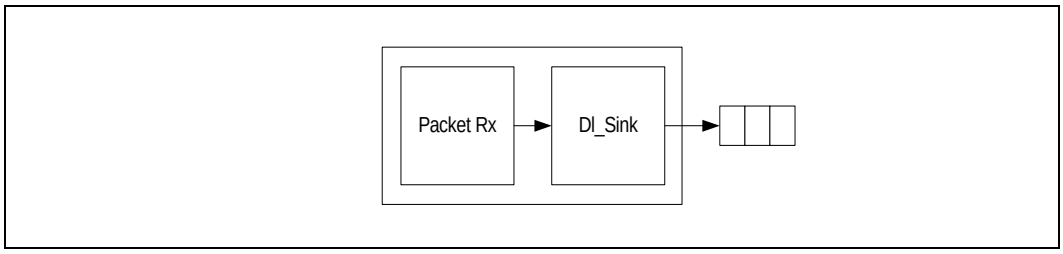


Figure 14-4. Dispatch Loop for the IPv4 Functional Pipeline

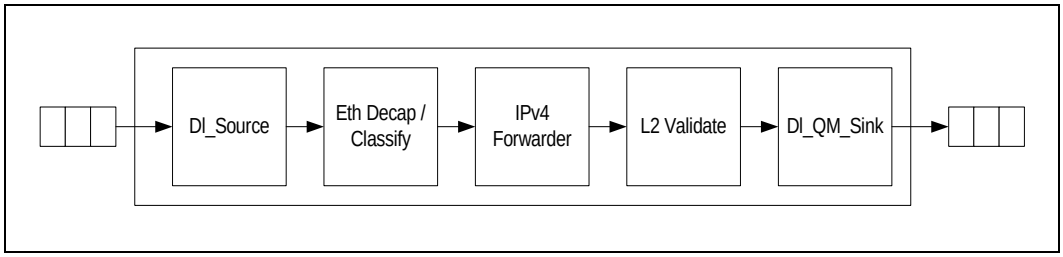
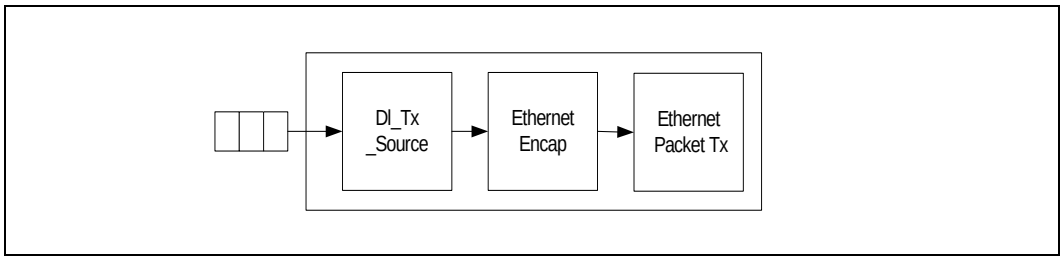


Figure 14-5. Dispatch Loop for the Ethernet Transmit Stage



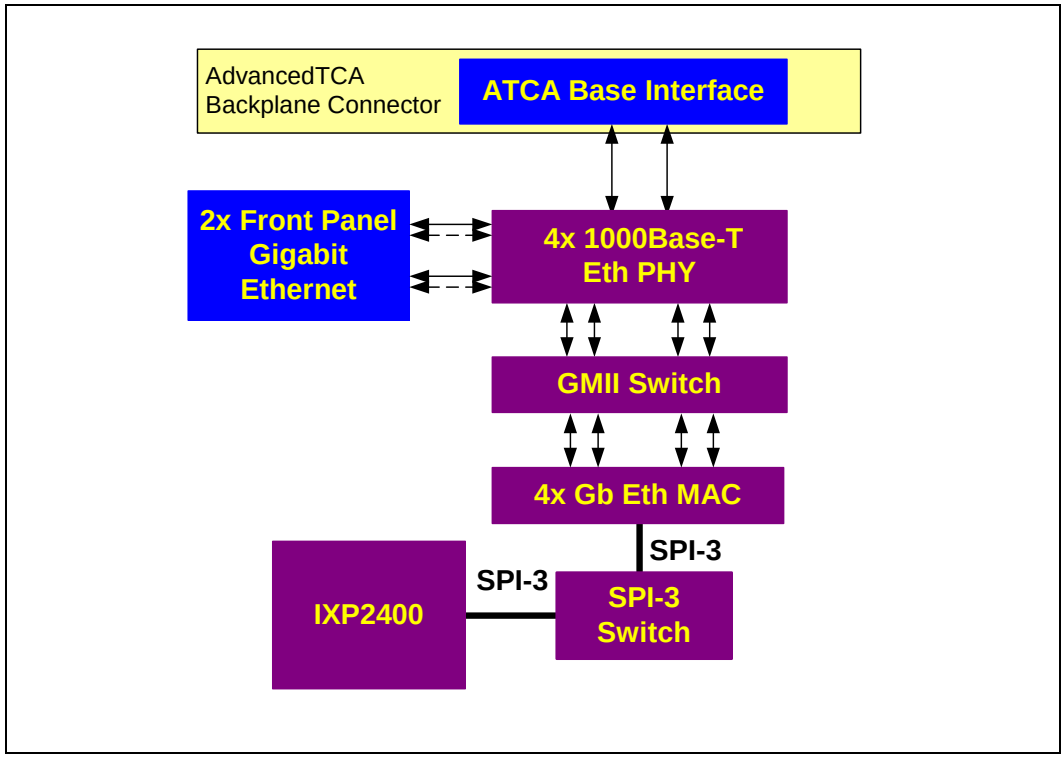
14.2.3 HW Architecture-Specific Code

14.2.3.1 Quad Gigabit Ethernet MAC Driver

A Quad Gigabit Ethernet MAC device is used to access front panel Gigabit Ethernet ports as well as the base interface in the backplane access module. Two ports are configured to support front panel ports; two other ports are configured for access to the base interface on the ATCA backplane.

The Quad Gigabit Ethernet MAC device is connected to the IXP2400 through SPI-3 buses for packet transmission/reception and through the slow port for device configuration and management. The device is configured for 1 Gigabit per second full duplex and its device driver supports only GMII (copper) mode. The Quad Gigabit Ethernet MAC device is also used for programming the Quad Ethernet PHY.

Figure 14-6. Ethernet Interface Connections to Quad Gigabit Ethernet MAC Device



14.2.3.2 Ethernet PHY Driver

The Quad Gigabit Ethernet PHY is connected to the Quad Gigabit Ethernet MAC device through GMII switch. Two PHY ports are connected to front panel Gigabit Ethernet ports; two other ports are connected to the base interface on ATCA backplane. The Quad Gigabit Ethernet PHY is configured and managed by the IXP2400 through the Quad Gigabit Ethernet MAC device.

14.3 Performance Characterization

The Intel® IXP2400 Network Processor operates at 600 MHz. For a minimum Ethernet packet of 64B, the packet inter-arrival time at 4 Gbps line rate is 100 microengine cycles. In order to maintain line rate for minimum packets, each stage of the pipeline cannot exceed this budget. In other words, each stage of the pipeline needs to retire a packet every 100 cycles. Table 14-2 summarizes the performance analysis for the Ethernet pipeline.

Table 14-2. Performance Characterization for the Ethernet Pipeline

Line rate for 4 Gig ports	4 Gigabits/sec
Min Ethernet packet size	64 bytes (+ 20 byte inter packet gap)
Packet throughput for minimum packets	5.95 million packets/sec = (4 / (84*8)) * (10**9)
IXP2400 clock frequency	600 MHZ
Inter-packet arrival time for minimum packets	600/5.95 = 100.84 cycles
Compute cycles per packet for a single microengine	100
Latency per packet for a context pipe single microengine	100 * 8
Compute cycles per packet for n microengines in parallel	100*n
Latency per packet for n microengines in parallel	100*8*n

14.4 System Resource Allocation

Table 14-3 shows the system resources mapped for the Intel® IXP2400 Network Processor. This mapping reflects the system defaults and may be changed to match the needs of a specific application. Microengine allocation has been made to optimize the performance of this specific pipeline application; it may be modified for other applications.

The physical assignment of function to microengine is important since it not only affects when the next neighbor registers and signaling can be utilized, but it also affects the utilization of the internal command bus and S-Push/Pull buses. Since ME0-ME3 belong to Microengine Cluster 0 and ME4-ME7 belong to Microengine Cluster 1, this assignment attempts to balance the usage of the command bus and S-Push/Pull buses across the two clusters.

The IXP2400 supports two SRAM channels and one DRAM channel. Table 14-4 shows the SRAM, DRAM, and scratch memory utilized for this application. These values are allocated by the Intel XScale® core application and patched to the microcode upon startup. The values are defined in a system XML configuration file `ix_sa_registry.xml` and may be changed as required.

Table 14-3. System Resources Mapped for the IXP2400

Microblock	ME #	Communication
Packet Rx	ME0	Auto-push status from MSF
IPv4 Forwarder + Eth Decapsulation/ Classifier + L2 Validation	ME1, ME2, M7	Scratch Ring
Queue Manager	ME3	Scratch Ring
Packet Scheduler	ME4	Scratch Ring
Packet TX + Eth Encapsulation	ME5, ME6	Scratch Ring

Table 14-4. SRAM, DRAM and Scratch Utilization for System Resource Allocation

Item	Size per entry in bytes	Number of entries	Total SRAM used	Total DRAM used	Total Scratch used
Buffer descriptors	32	16000	512000	-	-
Buffers	2048	16000	-	32768000	-
Queue descriptors	16	1025	16400	-	-
Layer-2 table with mapping from next hop id to Ethernet header	16	1024	16384	-	-
Trie table	64B for a trie entry	1024 used for approximation	256kB+1KB+1/ 2*1024*64B= 295936	-	-
Route table (next hop information)	16	1024	-	16384	-
IPv4 statistics	32	16 (needed 4)	-	-	512
IPv4 directed broadcast table	32	256 + 32 extra	9216	-	-
Processing control block	4	1	4	-	-
Ring from packet RX to packet processing pipeline (IPv4+Eth Decap/ Classify)	5*4B=20	204	-	-	1024*4B= 4096
IPv4 to QM ring	3*4B=12	170	-	-	512*4= 2048
Scheduler to QM ring	1*4B=4	512	-	-	512*4= 2048
QM Q-Array entries	4	16	64	-	-
Buffer free list Q-Array entry	4	4	16	-	-
uCode to Xscale core priority rings (1 AF+ 1 BE)	2*4B	64	-	-	2*512 = 1024
Xscale core to uCode rings (1 CNTRL + 1 DATA)	1*4B, 2*4B	128	-	-	128*4+ 256*4= 1536
QM to packet TX rings	1*4B=4	256	-	-	4*256*4B= 4096

14.5 Microblock Interfaces

This section describes the interfaces between the different microblocks for this pipeline application. In most of the messages, there is a valid bit is used to prevent a value of zero from being enqueued on the scratch ring. Zero is used to detect a case where the scratch ring is empty. So the valid bit helps distinguish between a zero value that was actually enqueued versus a case where the ring is empty.

14.5.1 Packet RX and Packet Processing Microengines

The interface between Packet RX and Packet Processing Microengines is identical to the OC-12 POS/Ethernet IPv4 Forwarding Application – see Section 13.5.1, “Packet RX and Packet Processing Microengines” on page 193.

14.5.2 Packet Processing Microengines and Packet QM

The interface between Packet Processing Microengines and Packet QM is identical to the OC-12 POS/Ethernet IPv4 Forwarding Application – see Section 13.5.2, “Packet Processing Microengines and Packet QM” on page 194.

14.5.3 Packet Scheduler and Packet QM

The interface between Packet Scheduler and Packet QM is identical to the OC-12 POS/Ethernet IPv4 Forwarding Application – see Section 14.5.3, “Packet Scheduler and Packet QM” on page 206.

14.5.4 Packet Queue Manager and Packet TX

The interface between the Packet Queue Manager and the Packet Transmit microengines is four scratch rings—one per Gigabit Ethernet port. Table 14-5 describes each entry in the scratch ring—which is one word.

Table 14-5. One-word Scratch Ring (Packet Queue Manager and Packet TX)

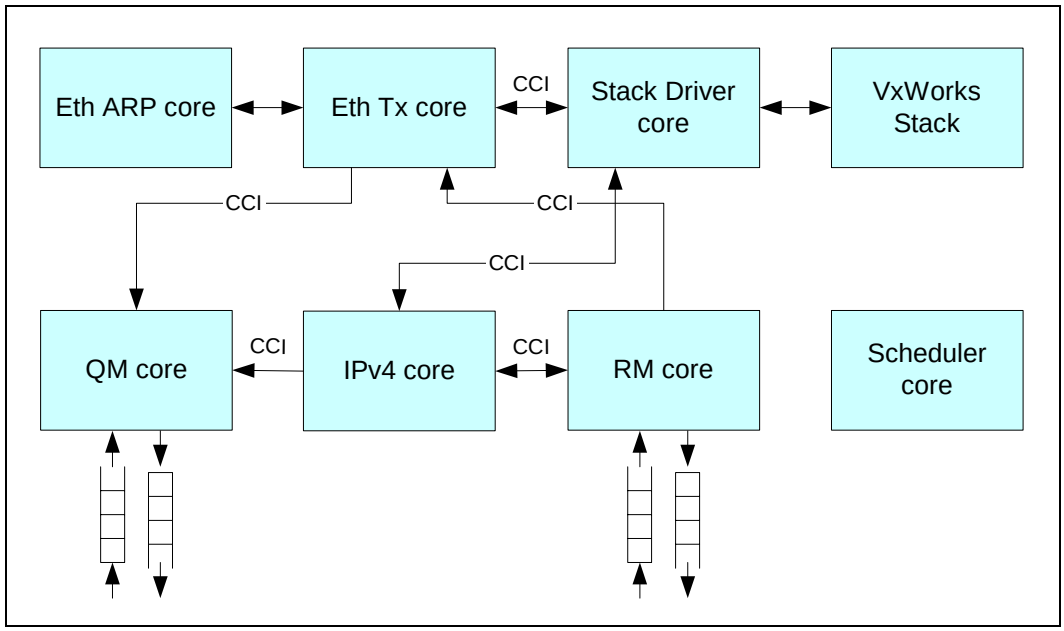
LW	Bits	Size	Field	Description
0	31	1	Valid Bit	Must be 1
	30:28	3	Reserved	Reserved
	27:24	4	Port Number	Port number
	23:0	24	SOP Buffer Descriptor	Pointer to SOP buffer descriptor in SRAM in long words (same as bits 23:0 of buffer handle)

14.6 Core Component Usage

The Quad Gigabit Ethernet Forwarding pipeline application uses standard core components customized to use only channel 0 for SRAM. Figure 14-7 shows the interconnection between the pipeline application’s core components. Resource Manager and Queue Manager core components

employ scratch rings for communication with microblocks on microengines. Interactions between IPv4, Ethernet Tx, Stack Driver, Resource Manager and Queue Manager are managed by the Core Component Interface (CCI).

Figure 14-7. Core Components in the Quad Ethernet IPv4 Forwarding Application



ATM/Ethernet IPv4 Forwarding Application for IXDP24X1 15

This chapter describes an IPv4 Forwarding software application for Ethernet and ATM implemented on an Intel® IXP2400 Network Processor. It provides a high level design overview and lists the different software components used to build this application. The example application uses standard building blocks from the IXA SDK.

The application described in this chapter is supported on the Intel® IXDP 2401 Advanced Development Platform, which uses a single Intel® IXP2400 Network Processor installed on the Intel® IXMB2401 baseboard.

Note: It is important that all applications developed for the IXDP24X1 platform must have the IX_PLATFORM_2401 flag defined in the project makefiles, for both the core components and the microblocks. An example of required flag definitions may be found in the makefiles of this application. By default, newly created projects under the Windriver* Tornado* development environment have the flag defined as IX_PLATFORM_2400. For this application, the flag must be changed to IX_PLATFORM_2401.

15.1 Hardware Overview

The application runs on the Intel® IXDP 2401 Advanced Development Platform. Up to two media mezzanine boards may be connected to the baseboard. Two SRAM channels are required, so one additional QDR with 4MB memory must be used. The baseboard is equipped with two daughterboard connectors (DB1 and DB2). There are two available mezzanine boards.

- 4xOC-12 POS ATM mezzanine card
- 4x1Gigabit Ethernet mezzanine card

Table 15-1 presents all possible hardware configurations supported by the ATM/Ethernet IPv4 Forwarding Application for the Intel® IXDP 2401 platform.

Table 15-1. Supported Hardware Configurations

DB1	DB2	Description
4xOC-12 ATM	4x1GE	one Gigabit Ethernet port available, one OC-12 ATM port. The ATM-IPv4-ETH application supports only one OC-12 ATM interface.

The example application receives traffic from the Ethernet media interface and transmits to the corresponding VCC on the ATM media interface with the configured encapsulation (VC MUX/ LLC SNAP). It also receives traffic from the ATM media interface and forwards it to the Gigabit Ethernet interface adding correct MAC addresses.

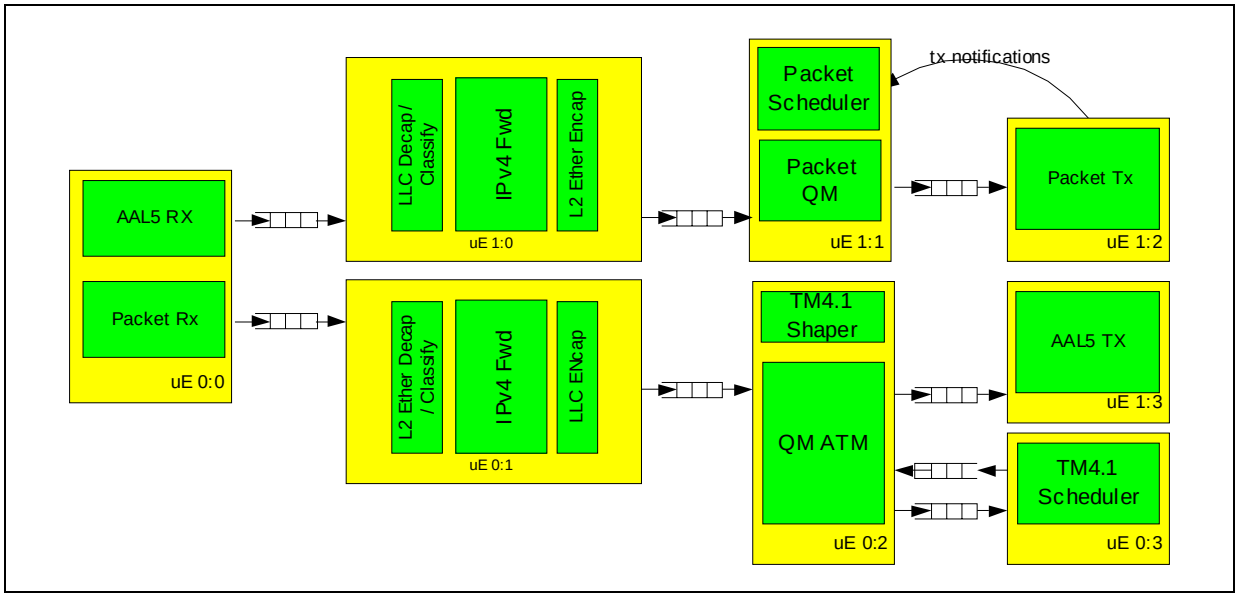
Note: Packet forwarding between two VCs is not supported in this release.

15.2 Software Overview

Figure 15-1 shows the microblocks needed to implement an OC-12 ATM/Ethernet IPv4 forwarding application. Context pipe stages like AAL5 TX, Ethernet TX and ATM TM4.1 Scheduler occupy one single microengine. All other pipe stages must work simultaneously. For example, AAL5 RX and Ethernet RX work on one single microengine, QM Packet and Scheduler Packet work on one single microengine, and the QM ATM and QM Shaper work on one single microengine. The forwarder consists of the IPv4 microblock. One microengine is dedicated to forwarding from the ATM media interface to the Ethernet media interface and one microengine is dedicated to forwarding from the Ethernet to the ATM media interface.

The packet processing blocks include the IPv4 Forwarder, the LLC decapsulation/classify microblock, the L2 Ethernet decapsulation/classify microblock, the LLC SNAP encapsulation and the L2 Ethernet encapsulation microblock. The ATM interface supports VC MUX and LLC encapsulation. There are eight microengines that run in parallel and execute the packet processing code.

Figure 15-1. ATM-IPv4-Ethernet Application Microblocks



The ATM/Ethernet IPv4 Forwarding application can be tested on the hardware platform using the Core Components and the transactor platform. The IXA SDK contains test configurations and packet streams for running on hardware and under simulation. For details, see the Readme file for the ATM/Ethernet IPv4 Forwarding Application.

Two encapsulation types are supported: VC multiplexing and LLC encapsulation. Virtual Connections that supports VC multiplexing carry only one specific type of traffic. For example, in the case of IPv4, traffic classification is determined by VC configuration. Virtual Connections that support LLC encapsulation can carry many different traffic types and the packet classification is determined from packet content. Supported encapsulations are listed below:

- ATM, VC multiplexing

```
IPv4 ([RFC2684])
C-Data /* IPv4 */
```

- ATM, LLC encapsulation (options determined from packet contents)

```
Routed packets
PPP NLPID ([RFC2364], [RFC1661])
LLC=FEFE03, NLPID=CF, PID= 0021, C-Data, PAD  /* IPv4 */
Routed SNAP ([RFC2684])
LLC=AAAA03, OUI = 000000, PID= 0800, C-Data  /* IPv4 */
```

Note: The ATM/Ethernet IPv4 Forwarding application supports only the "fast" path. Currently there is no support for communication between the microengines and Intel XScale® core. The described application requires two separate free lists. This is because two different Queue Managers are used: (QM ATM and QM Packet) which must work on two separate free lists. The buffer management implementation does not offer fully functional support for two (or more) free lists. Buffers sent from microcode to core components cannot be classified and assigned to the correct free list.

15.3 Data Flow

The Intel® IXP2400 Network Processor receives ATM cells or Ethernet frames that carry IPv4 datagrams. The frames are assembled into IPv4 packets and the Layer-2 (Ethernet or PPP)/ATM cell headers are removed. Based on the IPv4 header, a Longest Prefix Match (LPM) lookup is performed and the packets are transmitted over the appropriate VC/port. The sections located below describe the data flow on the Intel® IXP2400 Network Processor.

15.3.1 AAL5 RX/Ethernet RX

The AAL5 RX and the Ethernet RX microblock work simultaneously on one microengine. Each interface type (ATM and Ethernet) is being serviced by four threads.

The Ethernet Rx microblock receives Ethernet frames from the MSF interface. The Ethernet Rx microblock typically receives a number of MPKTs per Ethernet frame (containing elements/ segments of Ethernet frames). Since jumbo Ethernet frames may be up to 9K bytes, some large packets may be stored in multiple buffers chained together as a link-list. The first MPKT of an Ethernet frame is processed further by an Ethernet Classify / Decap macro. Upon having received all of the MPKTs comprising a complete and valid Ethernet frame, buffer handles for the first and last packet in the chain are queued in the scratch ring.

The ATM AAL5 Rx microblock receives MPKTs containing ATM cells from the MSF interface. Each ATM cell is read into transfer registers. AAL5 ATM cells are reassembled into complete AAL5 PDUs. The ATM AAL5 Rx microblock hashes an ingress ATM cell's physical port, VPI, and VCI to perform a lookup yielding a VC flow ID. The VCID is an index to one of 64K possible VCs. Using the VCID, the ATM AAL5 Rx microblock fetches the VC information containing the current AAL5 PDU reassembly context. If the PDU reassembly context indicates the first cell of an AAL5 PDU, then a packet buffer is allocated for the AAL5 PDU, packet metadata is established from relevant VC information, and the ATM Classify/Decap macro is invoked to classify the AAL5 PDU payload.

When the final cell of an AAL5 PDU is received, the PDU CRC is checked for validity, and the packet buffer descriptor length is adjusted to strip the AAL5 PDU padding and trailer. Upon receipt of a complete and valid AAL5 PDU, buffer handles for the first and last buffer of the packet, along with other metadata, are passed to the next microblock in the functional pipeline via the scratch ring.

15.3.2 Packet Processing

The Packet Processing is responsible for packet forwarding according to the IPv4 protocol. It occupies 2 microengines. The IPv4 protocol could be conveyed within various L2 encapsulations, so depending on the input port type (ATM or Ethernet) all layers must be decapsulated. The decapsulation is done in the LLC Decap or the L2 Ethernet Decap microblocks. Then if the packet contains an IPv4 header, it is passed to the IPV4 Forwarder microblock, otherwise it is dropped. Currently communication between microengines and Intel XScale® core is not supported.

The IPv4 Forwarder microblock forwards IPv4 packets based on L3 addressing. The IPv4 Forwarder microblock uses a packet descriptor and accesses an IP header from the cache in the transfer registers. The IP packet is then validated against [RFC1812] and [RFC2644] within the data plane. If the IP packet fails any of the validation checks, the packet is dropped. The packet's IP header TTL is decremented, and the IP header checksum is updated accordingly. The packet's next hop is then determined (that is, the next destination to which the packet is forwarded). To do that, the IP packet's destination address is passed to a 5-trie Longest Prefix Match (LPM) algorithm that yields a next hop index, which is used to obtain the next hop information. The information includes the output port and next hop ID, which is subsequently used to access the outgoing link layer information. The packet metadata is updated with the next hop ID. If the 5-trie algorithm fails (the best match cannot be determined), the packet is dropped.

After the IPv4 Forwarding the packets must be encapsulated with an L2 header or VC MUX/LLC header.

15.3.3 Packet-Based Queue Manager

The Packet-Based Queue Manager (QM) performs enqueue/dequeue operations on the hardware-assisted SRAM queues for packet-type traffic. The QM receives enqueue requests from the IPv4/DiffServ pipeline through a scratch ring. Another scratch ring is fed with dequeue requests from the packet scheduler. When the queue state changes between empty and non-empty, QM sends a transition message to the scheduler (via next neighbor registers). After every dequeue operation, the QM passes a transmit request to the scratch ring served by a Tx microblock. Dequeue requests come from the packet scheduler microengine.

15.3.4 Packet Scheduler

The Packet Scheduler selects constant-length packet segments to be transmitted to the MSF interface. The scheduler employs Round Robin (RR) among the fabric ports and Weighted Round Robin (WRR) among the port queues.

The Packet Scheduler sends the Queue Manager microblock a message to dequeue a packet from a specific port's queue. The Queue Manager microblock services the request, and deposits a packet descriptor from the requested queue into the output packet ring.

15.3.5 Cell-Based Queue Manager

The Cell-Based Queue Manager microblock enqueues packets and dequeues cells to/from an egress VC scheduling queue. Client microblocks send the Cell-Based Queue Manager microblock a message indicating the type of action to perform (enqueue or dequeue), and the ID of the queue (VCID) to or from which a packet or cell will be enqueued/dequeued. Upon a queue transition

from queue empty to non-empty, or non-empty to empty, the Cell Scheduler microblock is sent a message indicating the type of transition and VC queue to which it pertains. The messages inform the Cell Scheduler to start or stop scheduling a given VC.

15.3.6 TM 4.1 Shaper

The Shaper microblock determines transmission times for cells comprising an AAL5 PDU. It calculates cell transmission times to ensure that the transmission of VC cells does not violate the traffic contract established for the VC, and update/populate ATM cell transmit time queues. That is, the calendar queues the Cell Scheduler microblock services to schedule ATM cells for transmission at specific times.

15.3.7 TM 4.1 Cell Scheduler

The Cell Scheduler microblock requests the TM 4.1 Shaper to calculate the intended departure time for the cells comprising an AAL5 PDU. The Tx Shaper updates cell transmit time queues (within a calendar queue), which the Cell Scheduler queries/services. The Cell Scheduler notes the current time, and schedules cells within the current cell transmit time queue. ATM cells for a VC are scheduled for transmission per the calendar departure time. Providing the target port isn't blocked, a cell (48 bytes of ATM cell payload) from the current packet at the head of the VC's queue is scheduled for transmission.

To effect scheduling of individual ATM cells, the Cell Scheduler microblock sends the Queue Manager microblock a message to dequeue a cell for a specific VC. The Queue Manager microblock services the request, and deposits a cell descriptor from the requested VC queue into the output ring.

15.3.8 Ethernet Tx

The Ethernet Tx microblock transmits Ethernet frames via the MSF interface as one or more consecutive MPKTs (containing elements/segments of Ethernet frames). The Ethernet Tx microblock fetches a packet buffer handle (to access an upstream packet descriptor) from the egress packet ring (a scratch memory ring); the packet descriptor references the payload of an Ethernet frame. Using the supplied context, the Ethernet Tx microblock proceeds to transmit frame MPKTs out the egress port. Upon transmitting all MPKT frames, the packet buffer(s) is recycled.

15.3.9 ATM AAL5 Tx

The ATM AAL5 Tx microblock transmits MPKTs containing ATM cells to the MSF interface. Each ATM cell is an individual element/segment of an AAL5 PDU. The ATM AAL5 Tx microblock fetches a buffer handle (to access a downstream packet descriptor) from the egress packet ring (a scratch memory ring). The ATM AAL5 Tx microblock uses the ATM VC flow ID to fetch VC information that contains the VC's AAL5 PDU Tx context (which maintains the current PDU Tx byte count, AAL5 PDU CRC residue, etc).

In the case of the last cell of an AAL5 PDU, the ATM AAL5 Tx microblock appends AAL5 PDU padding as required, and updates the AAL5 CPCS-PDU trailer with the PDU length and PDU CRC. Upon having constructed a complete ATM cell, the cell is transmitted out the egress port (specified within the packet descriptor). When the last cell of an AAL5 PDU is transmitted, the VC's Tx context PDU length and PDU CRC values get reset, and the associated packet buffers get recycled.

15.4 Dispatch Loops

There are six microblock groups, organized as dispatch loops, used in this application. For more information on dispatch loops, refer to the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual* "Dispatch Loop" chapter.

- Dispatch Loop for the Ethernet Receive Stage, Figure 15-2
- Dispatch Loop for the ATM Receive Stage, Figure 15-3
- Dispatch Loop for the IPv4 Forwarder packet processing (ATM to Ethernet), Figure 15-4
- Dispatch Loop for the IPv4 Forwarder packet processing (Ethernet toATM), Figure 15-5
- Dispatch Loop for the Ethernet transmit stage, Figure 15-6
- Dispatch Loop for the ATM transmit stage, Figure 15-7

The Cell/Packet Queue Manager, Scheduler Packet TX, and TM4.1 blocks do not use a dispatch loop, though they still use the dispatch loop macros where required.

Note: The system microblocks `dl_source, dl_sink, dl_qm_sink,` etc. are application-specific. They may be changed for different packet processing pipelines.

Figure 15-2. Dispatch Loop for Ethernet Receive Stage

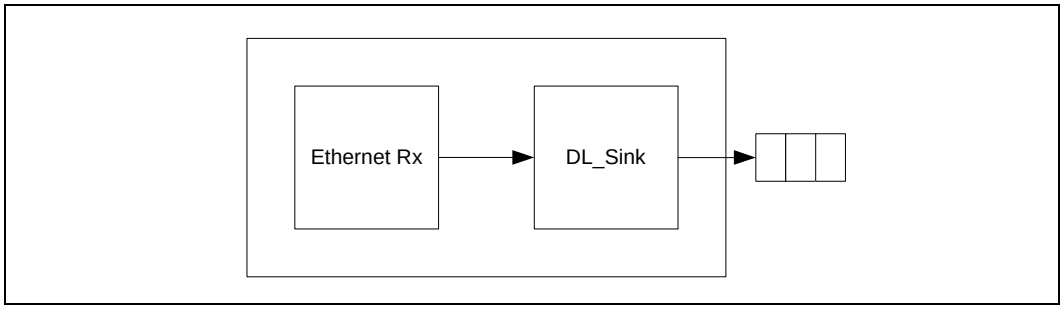


Figure 15-3. Dispatch Loop for ATM Receive Stage

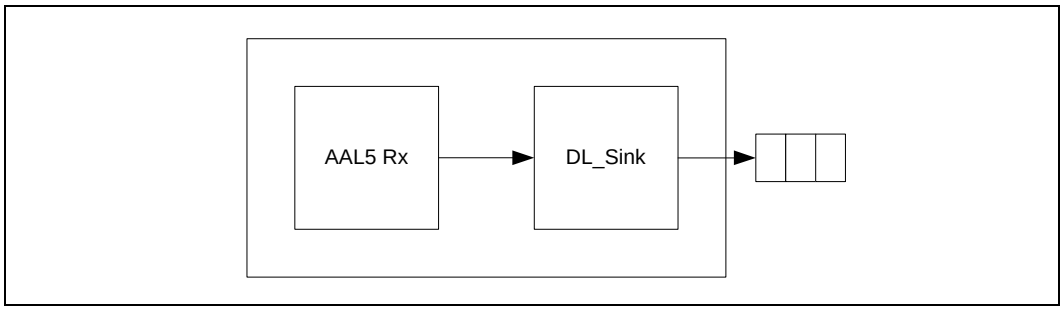


Figure 15-4. Dispatch Loop for IPv4 Forwarder Packet Processing (Ethernet to ATM)

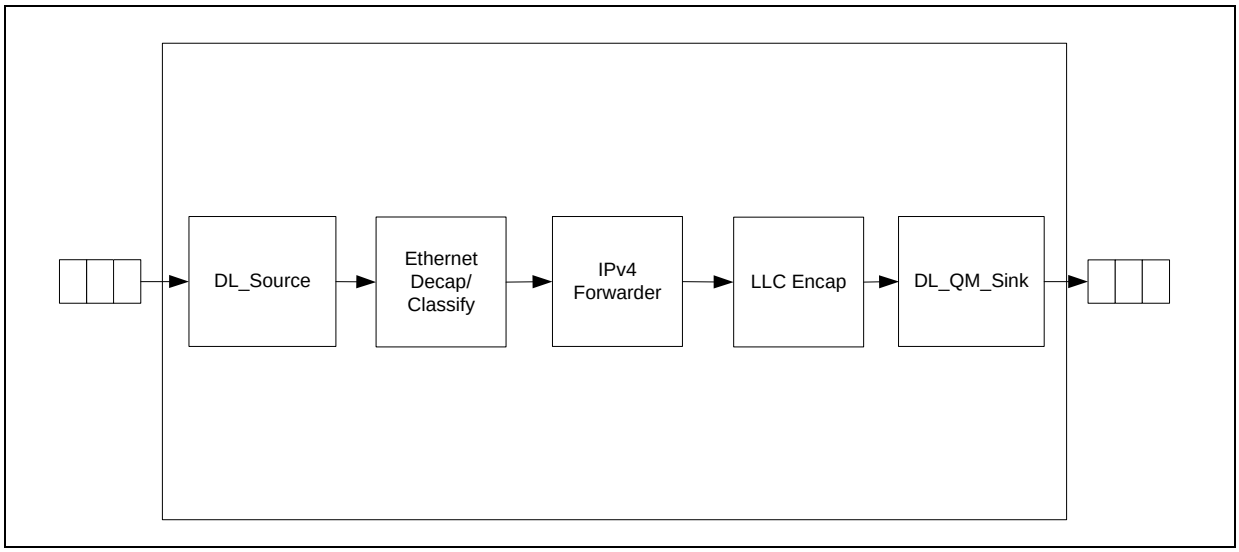


Figure 15-5. Dispatch Loop for IPv4 Forwarder Packet Processing (ATM to Ethernet)

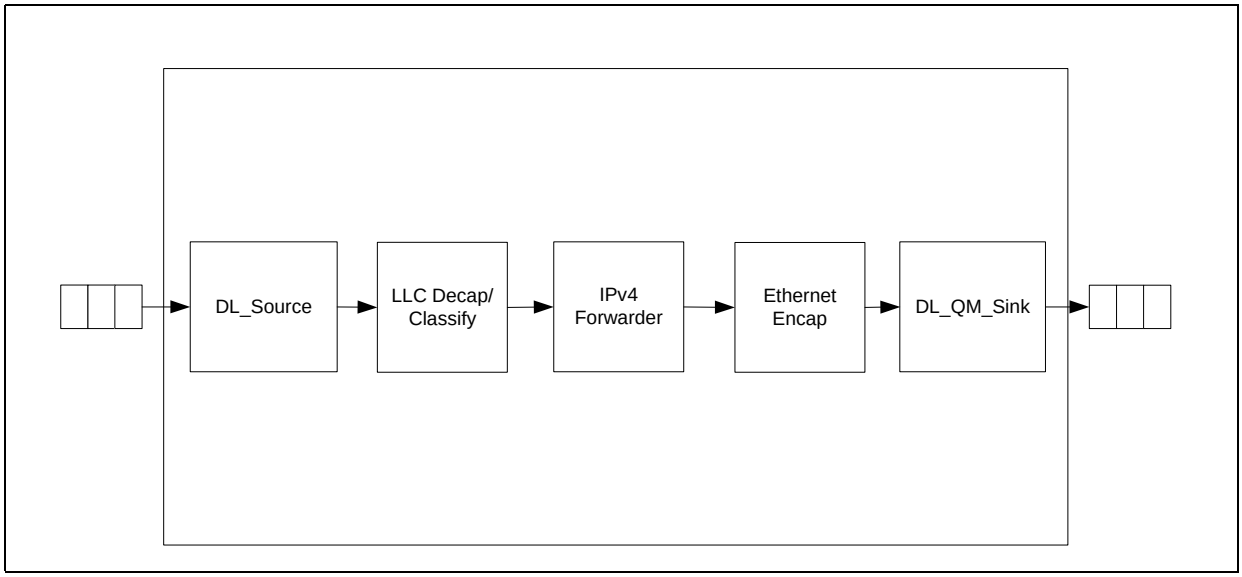


Figure 15-6. Dispatch Loop for ATM Transmit Stage

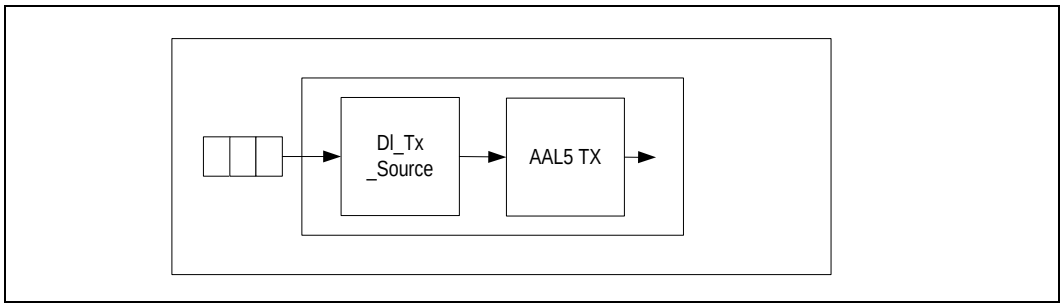
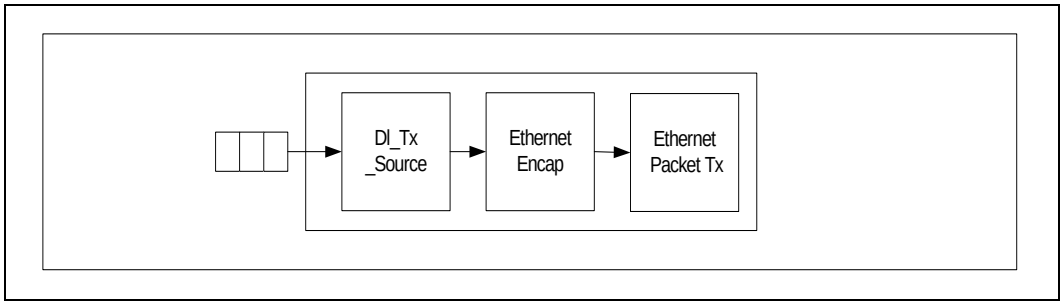


Figure 15-7. Dispatch Loop for Ethernet Transmit Stage



15.5 Performance Characterization

The Intel® IXP2400 Network Processor operates at 600 MHz frequency.

The OC-12 line rate is 622 Mbps, but the SONET overhead (approximately 3.8%) reduces it effectively to 599 Gbps available to ATM cells. An ATM cell payload with an ATM header forms a 53-byte cell. Assuming 53 bytes/cell: (53 bytes/cell * 8 bits/byte)/599 Mbps equals 708 ns/cell. At 600 MHz, this results in 425 cycles/cell.

Ethernet has variable-sized frames and a variable per-frame cycle budget. The worst case is minimum-sized 64-byte frames, so they are the focus for per-frame calculations here. A 64-byte frame actually occupies 84 bytes on the wire: (12 byte Inter Packet Gap) + (8 byte preamble) + (46 byte payload) + (14 byte Ethernet Header) + (4 byte Ethernet FCS) = 84 bytes/minimum frame} Assuming 84 bytes/frame: (84 bytes/frame * 8 bits/byte)/1 Gbps equals 672 ns/frame. At 600 MHz, this results in 403 cycles/frame.

For minimum Ethernet packets of 64 bytes in length and minimum POS packets of 49 bytes in length, the packet inter-arrival time at 6 Gbps line rate for Ethernet and 2.4 Gbps OC48 line rate for POS is 91 microengine cycles. In order to maintain line rate for minimum length packets, each stage of the pipeline cannot exceed this budget. In other words, each stage of the pipeline needs to retire a packet every 91 cycles.

Table 15-2 summarizes the performance analysis for the pipeline.

15.6 System Resource Allocation

Table 15-2 shows the system resources mapped for the Intel® IXP2400 Network Processor. This mapping reflects the system defaults and may be changed. The allocation of microengines is done such that it optimizes the performance of this specific application and may be changed for other applications.

Table 15-2. System Resources Mapped for the Intel® IXP2400 Network Processor

Microblock	ME #	Communication
Ethernet Rx/AAL5 RX	ME00	Auto-push status from MSF
L2 Ethernet Decap + IPv4 Fwd + LLC Decap + LLC Encap + L2 Encap	ME01, ME10	Scratch Ring
QM Shaper	ME02	Scratch Ring
TM4.1	ME03	Scratch Ring
Ethernet TX	ME12	Scratch Ring
AAL5 TX	ME13	Scratch Ring
QM Packet + Scheduler Packet	ME11	Scratch Ring

Table 15-3. SRAM Memory Map

Table Name	Size [bytes]	SRAM Channel 0 Usage	SRAM Channel 1 Usage
AAL5 RX hash table (primary)	16	4096*16	
AAL5 RX hash table (secondary)	32	4096*32	
AAL5 RX VC Info	64	4096*64	
AAL5 Port statistics	32	4096*32	
AAL5 TX statistics (cells per outport)	4	4*4	
AAL5 TX statistics (packets per outport)	4	4*4	
AAL5 TX statistics (cells per VCC)	4	4096*4	
AAL5 TX statistics (packets per VCC)	4	4096*4	
AAL5 TX Context table	64	4096*64	
GCRA table	64	4096*64	
PortShaping table	4		4*4
Portinfo Table	64		4*4
UBR TQ Table	128		4096*128
nrtVBR TQ Table	128		4096*128
rtVBR TQ Table	128		4096*128
HBR TQ Table	128		4096*128
LLC next hop table	128		4096*128
Next Hop Table	8	4096*8	
† Compiled optionally – not for benchmarking			

Table 15-3. SRAM Memory Map (Continued)

Table Name	Size [bytes]	SRAM Channel 0 Usage	SRAM Channel 1 Usage
QM Q-Array entries	4	16*6	
Packet RX statistics†	32		512
Packet TX statistics†	16		
I/O Buffer Descriptors	32	8192*32	8132*32
Total	n/a	1441920	2882208
† Compiled optionally – not for benchmarking			

15.7 Microblock Interfaces

This section describes the interfaces between the different microblocks for this pipeline application. In most of the messages, there is a valid bit is used to prevent a value of zero from being enqueued on the scratch ring. Zero is used to detect a case where the scratch ring is empty. So the valid bit helps distinguish between a zero value that was actually enqueued versus a case where the ring is empty.

15.7.1 Common RX to Packet Processing

The interface between the Common Receive microblock (AAL5 RX and Ethernet Rx) and the Packet Processing microengines (IPv4 Forwarder) is a scratch ring. Table 15-4 describes each entry in the scratch ring—which is five words.

Table 15-4. Common RX to Packet Processing Microengines Scratch Ring Interface

Variable	Size [bits]	Description
buff_handle	32	A handle to a buffer
buff_handle_eop	32	A handle to the last buffer in buffer chain – or NULL if single-buffer packet.
buffer_offset	16	The offset of the packet in the first buffer
buffer_size	16	Size of data in the first buffer
packet_size	16	Size of the whole packet
free_list	4	Freelist ID
rx_stat	4	Receive status flags
header_type	8	A packet type: ETHER_TYPE, PPP_TYPE
input_port	16	Input port number
output_port	16	Output port number – here unused – always 0

15.7.2 Packet Processing to Packet Queue Manager

The interface between the Packet Processing microengines (IPv4 Forwarder) and Packet QM is a scratch ring. Table 15-5 describes each entry in the scratch ring— which is five words.

Table 15-5. Packet Processing to Packet Queue Manager Scratch Ring Interface

Variable	Size [bits]	Description
buff_handle	32	A handle to a buffer
buff_handle_eop	32	A handle to the last buffer in buffer chain – or NULL if single-buffer packet.
validity bit	1 [31]	If set message is valid – prevention from producing value 0 on the ring
queue_number	31	output_port * 16 + class_id

15.7.3 Scheduler to Queue Manager

The interface between the Packet Scheduler and the packet-based Queue Manager is a Scratch Ring.

Table 15-6. Scheduler to Queue Manager Scratch Ring Interface

Variable	Size [bits]	Description
validity bit	1 [31]	If set message is valid – prevention from producing value 0 on the ring
Queue_number	31	output_port * 16 + class_id

15.7.4 Queue Manager to Scheduler

The interface between the packet-based Queue Manager and the Packet Scheduler is a Next Neighbor Ring.

Table 15-7. Queue Manager to Scheduler Next Neighbor Ring Interface

Variable	Size [bits]	Description
Validity bit	1 [31]	If set message is valid – prevention from producing value 0 on the ring
Queue_number	31	output_port * 16 + class_id

15.7.5 Queue Manager to Packet TX

The interface between the packet-based Queue Manager and the Packet Tx blocks is a Scratch Ring.

Table 15-8. Queue Manager to Packet TX Scratch Ring Interface

Variable	Size [bits]	Description
Output_port	8	Output port number
buff_handle	24	A handle to a buffer without SOP and EOP flags (the highest byte conveys ouput_port)

15.7.6 Queue Manager to TM 4.1 Shaper

The interface between the cell-based Queue Manager and the Schaper blocks is a next neighbor ring.

Table 15-9. Queue Manager to TM 4.1 Shaper Next Neighbor Ring Interface

Variable	Size [bits]	Description
Valid bit	1 [31]	The enqueue word is valid only if this bit is set
transition	1 [30]	Notification that queue has gone from empty to nonempty
CLP	1 [29]	
cell_count	11	Cell count provides the number of cells in the frame.
SOP	1 [17]	This field is important only for VCs shaped using GFR: “1” for Enqueue message when transition bit is also set, otherwise “0”
Enq VCQ	17	Queue Number that was enqueued
Valid bit	1 [31]	Must be 1
Transition	1 [30]	Notification that queue has gone from non-empty to empty
CLP	1 [29]	
cell_count	11	Cell count provides the number of cells in the frame.
SOP	17	This field is important only for VCs shaped using GFR:”1” for dequeue message when the Queue Manager has transmitted last cell from current packet and there ia another packet is the queue, otherwise “0”
Deq VCQ	17	Queue Number that was dequeued

15.7.7 TM4.1 Scheduler to Queue Manager

The interface between the TM4.1 Scheduler and the cell based Queue Manager blocks is a scratch ring.

Table 15-10. TM4.1 Scheduler to Queue Manager Scratch Ring Interface

Variable	Size [bits]	Description
Valid bit	1 [31]	Must be 1
Reserved	1 [30]	Reserved
Port	11	Output port number
CLP	1 [18]	
Reserved	1 [17]	Reserved
VCQ	17	Queue Number

15.7.8 Queue Manager to TM4.1 Scheduler

The interface between the cell-based Queue Manager and TM4.1 scheduler blocks is a scratch ring.

Table 15-11. Queue Manager to TM4.1 Scheduler Scratch Ring Interface

Variable	Size [bits]	Description
Valid bit	1 [31]	Must be 1
Reserved	1 [30]	Reserved
Port	11	Output port number
CLP	1 [18]	
Reserved	1 [17]	Reserved
VCQ	17	Queue Number
Buff_handle	32	Buffer Handle currently being transmitted for queue

15.7.9 Queue Manager to AAL5 TX

The interface between the cell based Queue Manager and AAL5 TX blocks is a scratch ring.

Table 15-12. Queue Manager to AAL5 TX Scratch Ring Interface

Variable	Size [bits]	Description
Valid bit	1 [31]	Must be 1
Reserved	1 [30]	Reserved
Port	11	Output port number
Reserved	3	Reserved
qnum	16	Queue Number
Buff_handle	32	Buffer Handle currently being transmitted for queue

POS/Ethernet IPv4 Forwarding Application for IXDP28x1 16

This chapter describes an IPv4 Forwarding software application for Ethernet and Packet over SONET (POS) implemented on one Intel® IXP2800 Network Processor. The chapter provides a high level design overview and lists the different software components used to build this application. This chapter describes the application in the context of Ethernet and POS media interfaces.

The application described in this chapter is supported on the Intel® IXDP28X1 Advanced Development Platform, which uses a single Intel® IXP2800 Network Processor.

This chapter focuses only on the fast path or microengine components of the design. The Intel XScale® core components for this application are described in the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*.

Note: It is important that all applications developed for the IXDP28X1 platform must have the IX_PLATFORM_2801 flag defined in the project makefiles, for both the core components and the microblocks. An example of required flag definitions may be found in the makefiles of this application. By default, newly created projects under the Windriver* Tornado* development environment have the flag defined as IX_PLATFORM_2800. For this application, the flag must be changed to IX_PLATFORM_2801.

16.1 Hardware Overview

The Intel® IXDP28X1 Advanced Development Platform consists of the IXMB28X1 baseboard, which is equipped with two daughterboard connectors (DB1 and DB2). Up to two media mezzanine boards (also called line cards) may be connected to the baseboard. There are three available mezzanine boards:

- 4xOC-12 POS ATM mezzanine board
- 1xOC-48 POS ATM mezzanine board
- 4x1Gigabit Ethernet mezzanine board

Table 16-1 presents all possible hardware configurations supported by the POS/Ethernet Ipv4 Forwarding Application for IXDP28X1.

Table 16-1. Supported Hardware Configurations

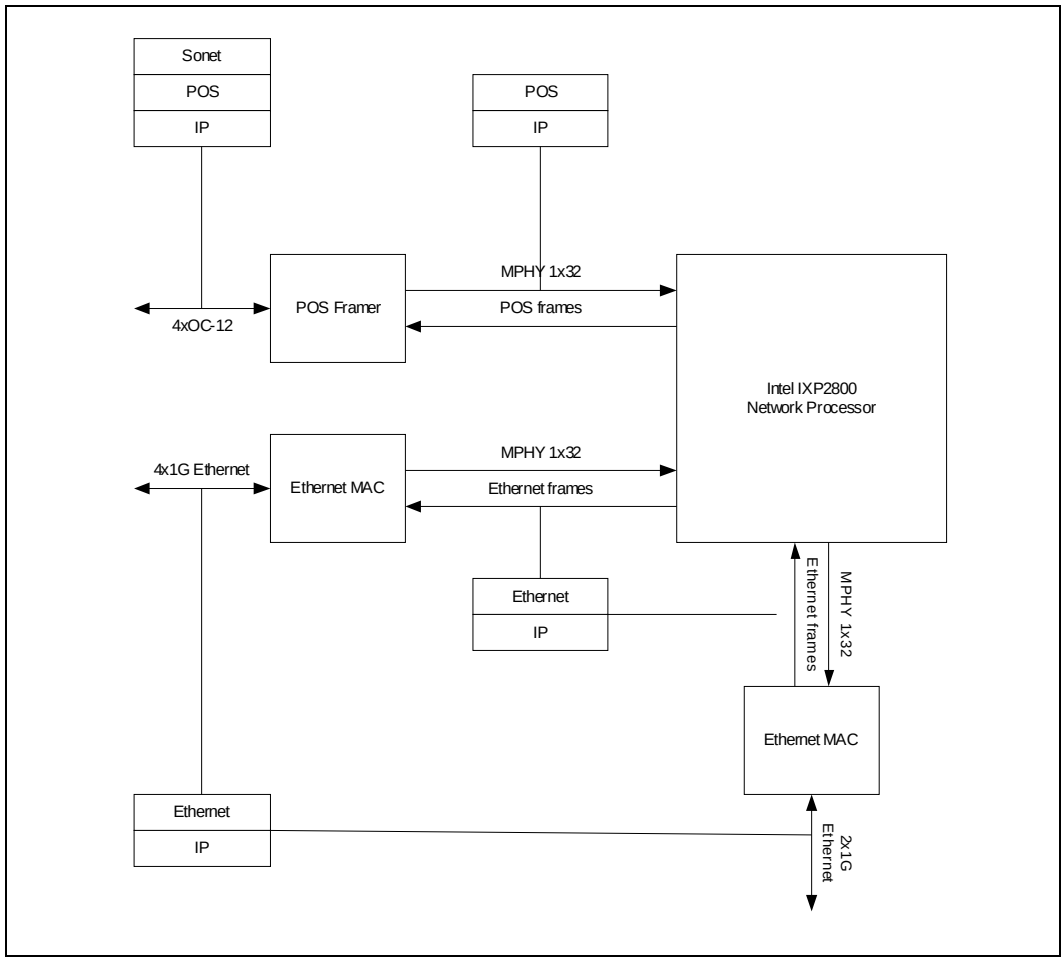
Backplane	DB1	DB2	Description	Throughput
2x1GE†	4xOC-12 POS	4x1GE	All supported ports available.	6.5 Gbps example: • 2x1GE 100% • 4xOC-12 100% • 2x1GE 100%
2x1GE†	4xOC-12 POS		POS on the front side and Ethernet on the backplane.	3.3 Gbps example: • 2x1GE 100% • 3xOC-12 100%
2x1GE†		4x1GE	Only Ethernet ports vailable.	Full bandwith
2x1GE†	1xOC-48 POS	4x1GE	1 POS on the front and all Ethernet ports supported	6.5 Gbps example: • 2x1GE 100% • 1xOC-48 100% • 2x1GE 100%
2x1GE†	1xOC-48 POS		POS on the front side and Ethernet on the backplane.	3.3 Gbps example: • 1x1GE 100% • 1xOC-48 100%
† 2x1Gigabit Ethernet base ATCA interfaces				

Figure 16-1 shows an Intel® IXP2800 Network Processor in a typical configuration. In this configuration, the IXP2800 is identified as the network processor. It receives traffic from the Ethernet or POS media interface and transmits to the other Ethernet or POS media interface.

The target hardware comprises up to ten physical media interfaces. A POS media mezzanine card installed on a baseboard provides four OC-12 interfaces. Four Gigabit Ethernet interfaces are provided on a Gigabit Ethernet mezzanine and two Gigabit Ethernet interfaces are available on the baseboard Backplane Access module.

The Intel® IXP2800 Network Processor receives POS or Ethernet frames that carry IPv4 datagrams. The frames are assembled into IPv4 packets and the Layer-2 (Ethernet or PPP) headers are removed. Based on the IPv4 header, a Longest Prefix Match (LPM) lookup is performed and the packets are transmitted over the appropriate port.

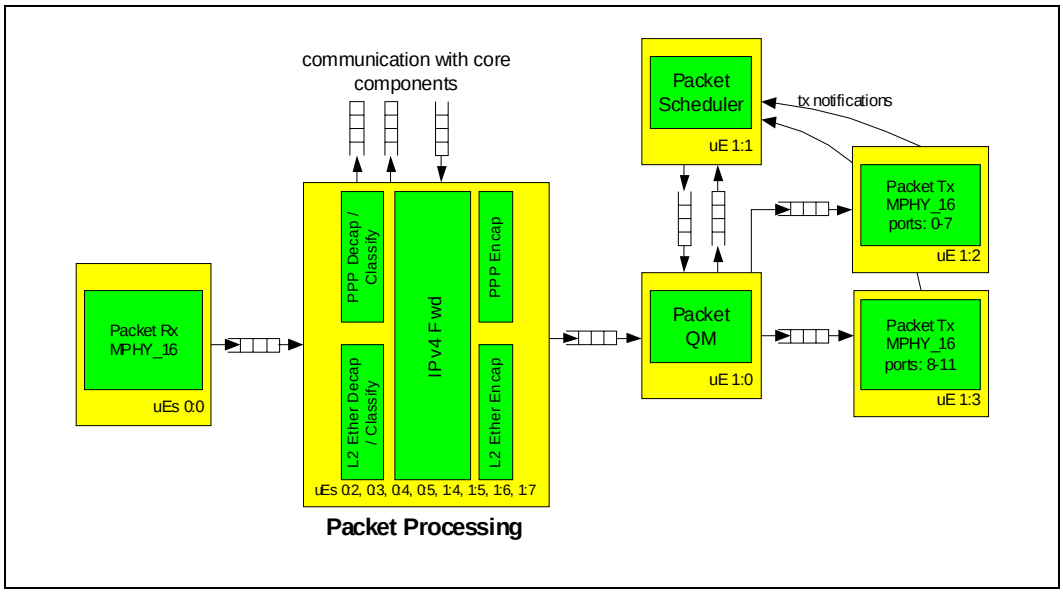
Figure 16-1. POS/Ethernet Hardware Configuration on IXDP28X1



16.2 Software Overview

Figure 16-2 shows the microblocks needed to implement an POS/Ethernet IPv4 forwarding application. All the context pipe-stages (for example, Packet Rx, Queue Manager, and Scheduler) occupy an entire microengine. Each context pipe-stage is mapped to a single microblock running on a microengine with or without a dispatch loop. The packet processing runs on eight microengines and implements decapsulation (Ethernet and PPP) together with encapsulation and the IPv4 forwarder blocks.

Figure 16-2. POS-Ethernet IPv4 Application Microblocks



The design for the application shown in Figure 16-2 is based on the guidelines specified in the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual*. The driver microblocks (Receive, Transmit, Scheduler and Queue Manager) run on different microengines from the packet processing code. In this design, each driver block occupies an entire microengine. The packet processing blocks include the IPv4 Forwarder, the PPP decapsulation/classify microblock, the L2 Ethernet decapsulation/classify microblock, the PPP encapsulation and the L2 Ethernet encapsulation microblock. There are eight microengines that run in parallel and execute the packet processing code.

16.2.1 Data Flow

This section describes the data flow on the Intel® IXP2800 Network Processor.

16.2.1.1 Packet RX

Packet reception from the MSF interface is done in the Packet RX microblock that runs on one microengine. It is a standard microblock compiled to run in MPHY_16 mode. The microblock performs packet reassembly from the incoming micropackets being burst on the SPI-4 BUS. Packets for processing are conveyed in I/O buffers, so they are copied from MSF receive buffers (RBUFs) into DRAM memory, and also packet descriptors are initially filled in SRAM. Then packet buffer handles and some meta-data about the packets are passed via scratch ring to be processed in the Packet Processing block.

16.2.1.2 Packet Processing

The Packet Processing is responsible for packet forwarding according to the IPv4 protocol. It occupies 8 microengines. The IPv4 protocol could be conveyed within various L2 encapsulations, so depending on the input port type, L2 PPP or L2 Ethernet MAC layers must be decapsulated. The decapsulation is done in the PPP Decap or the L2 Ethernet Decap microblocks. Then if the packet

contains an IPv4 header, it is passed to the IPV4 Forwarder microblock, otherwise (in the case of ARP, for example) packets are transferred through an exception ring to the Core Components framework running on the Intel XScale® core. There the packets are processed by the slow path.

The IPv4 Forwarder microblock forwards IPv4 packets based on L3 addressing. The IPv4 Forwarder microblock uses a packet descriptor and accesses an IP header from the cache in the transfer registers. The IP packet is then validated against [RFC1812] and [RFC2644] within the data plane. If the IP packet fails any of the validation checks, the packet is dropped. The packet's IP header TTL is decremented, and the IP header checksum is updated accordingly. The packet's next hop is then determined (i.e., the next destination to which the packet is forwarded). To do that, the IP packet's destination address is passed to a 5-trie Longest Prefix Match (LPM) algorithm that yields a next hop index, which is used to obtain the next hop information. The information includes the output port and next hop ID, which is subsequently used to access the outgoing link layer information. The packet metadata is updated with the next hop ID, and the packet is handed off to the L2 Validation microblock. If the 5-trie algorithm fails (the best match cannot be determined), the packet is sent to the Intel XScale® core to complete the LPM procedure.

After the IPv4 Forwarding, the packets must be encapsulated with an L2 header. The encapsulation takes place in the PPP Encap or the L2 Ether Encap microblocks, depending on the output (destination) port of the packet. The packet buffer handle and some related meta-data are passed via scratch ring to the Queue Manager microblock.

16.2.1.3 Packet-Based Queue Manager

The Packet-Based Queue Manager (QM) performs enqueue/dequeue operations on the hardware assisted SRAM queues for packet-type traffic. The QM receives enqueue requests from the IPv4 microblock through a scratch ring. When the queue state changes between empty and non-empty, QM sends a transition message to the scheduler (via next neighbor registers). After every dequeue operation, the QM passes a transmit request to the scratch ring served by the Packet TX microblock. Dequeue requests come from the packet scheduler microengine.

16.2.1.4 Packet Scheduler

Packet Scheduler selects packets to be transmitted out of the MSF interface. The Packet Scheduler sends a message to the Queue Manager microblock to dequeue a packet from a specific port's queue. The Queue Manager microblock services the request, and deposits a packet descriptor from the requested queue into the output packet ring.

16.2.1.5 Packet TX

The Packet TX microblock transmits packets via the MSF interface as one or more consecutive micro-packets are being burst on the SPI-4 BUS. The Packet TX microblock runs on two microengines and supports MPHY_16 mode. Thus up to 16 Gigabit Ethernet or OC-12 POS ports are supported. The first microengine transmits packets destined to ports 0 to 7, the second microengine transmits packets destined to ports 8 to 15.

The microblock fetches a transmit request from a scratch ring. The transmit request is used to access the packet meta-data. Using the supplied meta-data, the microblock fragments the packet into micropackets and sends them out of the MSF. Upon transmitting all fragments, the packet buffer(s) is recycled.

The Packet TX microblock periodically updates the scheduler with information about how many packets have been transmitted. If the packets in flight for a particular port (packets scheduled but not transmitted) exceed a certain limit (which depends on the bandwidth supported by that port),

then the scheduler stops scheduling more packets for that port. This combination of queuing packets in local memory and keeping track of the packets in flight, helps prevent head-of-line blocking.

16.2.2 Dispatch Loops

There are two microblock groups, called dispatch loops, used in this pipeline application.

- Dispatch Loop for the Packet Frame Reassembly Stage, shown in Figure 16-3
- Dispatch Loop for the IPv4 Forwarder packet processing, shown in Figure 16-4

The Queue Manager, Scheduler and Packet TX blocks do not use a dispatch loop, although they use the dispatch loop macros where required.

For more information on dispatch loops, refer to the *Intel® Internet Exchange Architecture Portability Framework Developer's Manual* "Dispatch Loop" chapter.

Note: The system microblocks `dl_source, dl_sink, dl_qm_sink`, etc are application-specific. They may be changed for different packet processing pipelines.

Figure 16-3. Dispatch Loop for the Packet Frame Reassembly Stage

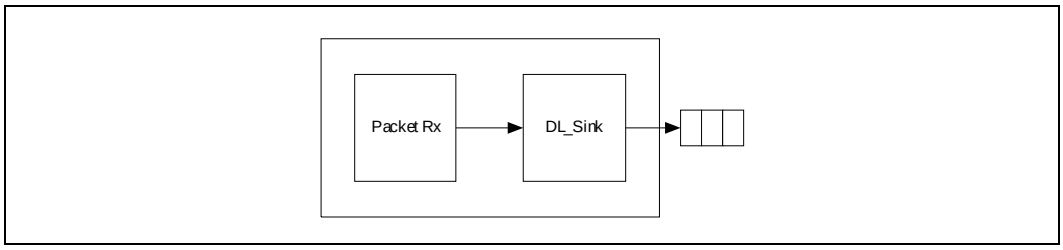
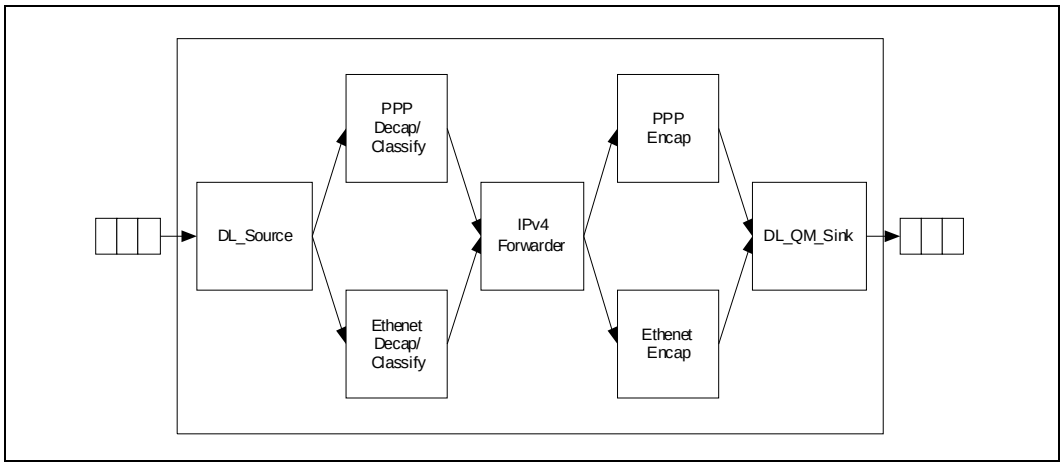


Figure 16-4. Dispatch Loop for IPv4 Forwarder Packet Processing



16.3 Performance Characterization

The Intel® IXP2800 Network Processor operates at 1400 MHz frequency. For a minimum Ethernet packets of 64 bytes in length and minimum POS packets of 49 bytes in length, the packet inter-arrival time at 6 Gbps line rate for Ethernet and 2.4 Gbps OC48 line rate for POS is 91 microengine cycles. In order to maintain line rate for minimum length packets, each stage of the pipeline cannot exceed this budget. In other words, each stage of the pipeline needs to retire a packet every 91 cycles. Table 16-2 summarizes the performance analysis for the pipeline.

Table 16-2. Performance Characterization for the POS-Ethernet IPv4 Application

Line rate for 6 Gigabit Ethernet Ports and 4 OC-12 POS Ports	8.54 Gigabits/sec
Min Ethernet packet size	64 bytes (+ 20 byte inter packet gap)
Packet Throughput for min Ethernet packets	8.93 million packets/sec = (6 / (84*8)) * (10^9)
Min POS packet size	49 bytes (40 byte TCP/IP, 2 bytes Address and Control, 2 byte PPP header, 4 byte FCS and 1 byte flag)
Packet Throughput for min POS packets	6.34 million packets/sec = (4/(49*8)) * 622 * (10^6)
Summarized Packet Throughput for all interfaces	15.3 million packets
Intel® IXP2800 Network Processor clock frequency	1400 MHZ
Inter-packet arrival time for min packets	1400/15.3 = 91.46 cycles
Compute cycles per packet for a single microengine	91
Latency per packet for a context pipe single microengine	91 * 8
Compute cycles per packet for n microengines in parallel	91*n
Latency per packet for n microengines in parallel	91*8*n

16.4 System Resource Allocation

Table 16-3 shows the system resources mapped for the Intel® IXP2800 Network Processor. This mapping reflects the system defaults and may be changed. The allocation of microengines is done such that it optimizes the performance of this specific application and may be changed for other applications.

Table 16-3. System Resources Mapped for the Intel® IXP2800 Network Processor

Microblock	ME #	Communication
Packet Rx	ME0	Auto-push status from MSF
L2 Rthernet Decap + L2 PPP Decap, +Ipv4 Fwd + L2 Ethernet Encap + L2 PPP Encap	ME2, ME3, ME4, ME5, ME12, ME13, ME14, ME15	Scratch Ring
Queue Manager	ME8	Scratch Ring
Scheduler	ME9	Scratch Ring
Packet TX Ports 0..7	ME10	Scratch Ring
Packet TX Ports 8..11	ME11	Scratch Ring

Table 16-4 shows data distribution in SRAM memory channels.

Table 16-4. SRAM Memory Map

Table Name	Size [bytes]	SRAM channel 0 usage	SRAM channel 1 usage	SRAM channel 2 usage	Comments
I/O Buffer Descriptors	32	512000		512000 †	16000 entries use 512000 bytes
Queue Descriptors	16	16400			1025 entries
L2 Table	16	2MB			64k entries
Trie Table			2MB		Structured tree of tries
Broadcast Table				8192	
Next Hop Table	8			4096	
QM Q-Array entries	4	64			
Buffer Free list Q-Array entry	4	16			
Packet RX statistics ‡	32		512		16 statistics
Packet TX statistics ‡	16			256	16 statistics

† Additional channel used when splitting buffer descriptors.

‡ Compiled optionally – not for benchmarking.

Table 16-5 shows the budget for every memory access that is needed for packet processing that influences the memory distribution.

Table 16-5. SRAM Channels Budget For Packets Processing with Minimal Length†

Microblock/Access	Operation	SRAM channel 0 utilization worst/best case	SRAM channel 1 utilization worst/best case	SRAM channel 2 utilization worst/best case	Comments
Packet RX/ I/O Buffer allocation	dequeue	4/4			
Packet Processing/ I/O buffer descriptor write	write	20/20 †4/4		-- †16/16	For min packets packet descriptor is written by Packet Processing when for packets > 128 bytes packet descriptor is also written by Packet RX.
Packet Processing/ IPv4 Directed Broadcast check	read			32/32	Worst case depends on the configuration – when there aren't conflicting directed broadcast hashes, it is the same as best case.
Packet Processing/ IPv4 lookup	read		20/12		Worst case tries to read up to 5 times
Packet Processing/ IPv4 NH read	read			8/8	
Packet Processing/ L2 Table read	read	16/16			
Queue Manager/ queuing	enqueue	4/4			

† Additional channel used when splitting buffer descriptors.

‡ Compiled optionally – not for benchmarking.

Table 16-5. SRAM Channels Budget For Packets Processing with Minimal Length†

Microblock/Access	Operation	SRAM channel 0 utilization worst/best case	SRAM channel 1 utilization worst/best case	SRAM channel 2 utilization worst/best case	Comments
Queue Manager/ QD read	read	12/0			In best case CAM always hits
Queue Manager/ dequeue	dequeue	4/4			
Packet TX/ I/O buffer descriptor read	Read	16/16 (4/4)		-- (12/12)	
Packet RX/Packet RX statistics†—number of packets RX	atomic increment		‡4/4		
Packet RX/Packet RX statistics†—number of bytes RX	atomic add		‡4/4		
Packet TX†—number of packets TX	atomic increment			‡4/4	
Packet TX/Packet TX statistics†—number of bytes TX	atomic add			‡4/4	

† Additional channel used when splitting buffer descriptors.

‡ Compiled optionally – not for benchmarking.

16.5 Microblock Interfaces

This section describes the interfaces between the different microblocks for this pipeline application. In most of the messages, there is a valid bit used to prevent a value of zero from being enqueued on the scratch ring. Zero is used to detect a case where the scratch ring is empty. The valid bit helps distinguish between a zero value that was actually enqueued versus a case where the ring is empty.

16.5.1 Packet RX to Packet Processing Microengine

The interface between the Packet Receive microblock and the Packet Processing microengines (IPv4 Forwarder + L2/PPP decap) is a scratch ring. Table 16-6 describes each entry in the scratch ring—which is five words.

Table 16-6. Packet RX to Packet Processing Microengine Scratch Ring Interface

Variable	Size [bits]	Description
buff_handle	32	A handle to a buffer
buff_handle_eop	32	A handle to the last buffer in buffer chain – or NULL if single-buffer packet.
buffer_offset	16	The offset of the packet in the first buffer
buffer_size	16	Size of data in the first buffer
packet_size	16	Size of the whole packet

Table 16-6. Packet RX to Packet Processing Microengine Scratch Ring Interface (Continued)

Variable	Size [bits]	Description
free_list	4	Freelist ID
rx_stat	4	Receive status flags
header_type	8	A packet type: ETHER_TYPE, PPP_TYPE
input_port	16	Input port number
output_port	16	Output port number – unused, always 0

16.5.2 Packet Processing to Queue Manager Microengine

The interface between the Packet Processing microengines (IPv4 Forwarder + L2/PPP decap + L2 Validate) and Packet QM is a scratch ring. Table 16-7 describes each entry in the scratch ring—which is five words.

Table 16-7. Packet Processing to Queue Manager Microengine Scratch Ring Interface

Variable	Size [bits]	Description
buff_handle	32	A handle to a buffer
buff_handle_eop	32	A handle to the last buffer in buffer chain – or NULL if single-buffer packet.
validity bit	1 [31]	If set message is valid – prevention from producing value 0 on the ring
queue_number	31	output_port * 16 + class_id

16.5.3 Scheduler to Queue Manager Microengine

The interface between the POS/Ethernet Scheduler and the packet-based Queue Manager is a Scratch Ring.

Table 16-8. Scheduler to Queue Manager Microengine Scratch Ring Interface

Variable	Size [bits]	Description
Validity bit	1 [31]	If set message is valid – prevention from producing value 0 on the ring
Queue_number	31	output_port * 16 + class_id

16.5.4 Queue Manager to Scheduler Microengine

The interface between the packet-based Queue Manager and the POS/Ethernet Scheduler is a Next Neighbor Ring.

Table 16-9. Queue Manager to Scheduler Microengine Next Neighbor Ring Interface

Variable	Size [bits]	Description
Validity bit	1 [31]	If set message is valid – prevention from producing value 0 on the ring
Queue_number	31	output_port * 16 + class_id

16.5.5 Queue Manager to Packet TX Microengine

The interface between the packet-based Queue Manager and the Packet Tx blocks is a Scratch Ring.

Table 16-10. Queue Manager to Packet TX Microengine Scratch Ring Interface

Variable	Size [bits]	Description
Validity bit	1 [31]	If set message is valid – prevention from producing value 0 on the ring
Output_port	7	Output port number
buff_handle	24	A handle to a buffer without SOP and EOP flags (the highest byte conveys ouput_port)

16.6 Core Components Integration

The POS/Ethernet Forwarding Application uses standard core components customized to use channels 0, 1, and 2 for SRAM. Figure 16-5 shows the interconnections between the application's core components. The Resource Manager and Queue Manager core components employ scratch rings for communication with microblocks on microengines. Interactions between IPv4, Ethernet Tx, POS Tx, Stack Driver, Resource Manager and Queue Manager are managed by the Core Component Interface (CCI).

Figure 16-5. POS-Ethernet IPv4 Application Core Components