

# **Symphony System Architecture Specification - Revision A**

This document is not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of the copyright holder.

Revision No.	Date	Author	Description of Changes
0.02	Dec. 27	Syed Shah	Feedback received from Sanjay
0.03	Dec. 29	Syed Shah	
0.04	Jan 3	Syed Shah	Added details on Credit Management, Virtualization
0.06	Jan 4	Mohammad	Added details on Power Management architecture
0.07	Jan 6	Syed	Updated Topology section
0.08	March 18	Sanjay Deshpande	Added Switch Architecture Chapter
0.09	April 3	John Coddington	Added Configuration Chapter
			Added Resiliency Chapter
			Added Interrupts & Errors Chapter
0.10	April 7	Dave Brown	Added Performance Monitoring and Debug Chapters; removed manual table of contents; added place-holder chapters for topics that were in the manual table of contents but had no corresponding chapter; basic cleanup
0.11	April 11	Sanjay Deshpande	Added section on Multicast Switches
0.12	April 12	John Coddington	Added section on Outside Spec
0.13	April 13	Syed & Shailendra	Added Packet Protocol, ATU and Credit Management
0.14	April 14	Syed & Dave	Massive cleanup; add section on VNs
0.50	April 25	All	Clean-up; released for external review as version 0.50 – hence jump is revision numbers from 0.14 to 0.50
0.51	May 5	All	Updated to use consistent terminology
0.52	July 21	John Coddington	Converted to FrameMaker
0.7 R1	Oct. 9	John Coddington	0.7 Review Copy Fixed Typos found in Memory Map Chapter
0.7 R2	Oct 11	John Coddington	Fixed Numbering Issue.
0.7	Oct 31	John Coddington	Final 0.7 Version

Revision No.	Date	Author	Description of Changes
0.9 R1	Feb 5	Shailendra Aulakh Darshan Alagud John Coddington Sanjay Despande Syed Shaw	0.9 Review Copy
0.9 R2	March 27	Shailendra Aulakh Darshan Alagud John Coddington Sanjay Despande Syed Shaw	0.9 Review Copy
0.9	April 3	John Coddington	Final 0.9 Version
0.95 R1	June 29	John Coddington Syed Shaw	0.95 Review copy for SW.
0.95 R2	Aug 18	Syed Shaw	0.95 Review copy
0.95 R3	Jun 19	John Coddington Syed Shaw	0.95 Review copy Added ATP Physical Layer, Multicast, Buffered Switch, Pressure and Starvation
0.95 R4	Sept 11	John Coddington	0.95 Review copy Integrated accumulated changes over past year



# Contents

## Terminology 15

### Introduction 21

2.1	Architectural Goals and Requirements .....	21
2.2	Layered Architecture .....	22
2.2.1	Transaction Layer .....	22
2.2.2	Network Layer .....	23
2.2.3	Data Link Layer (Packet Layer) .....	23
2.2.4	Physical Layer .....	23
2.3	Overall System Architecture .....	24
2.4	Topology .....	25
2.5	Quality of Service (QoS) .....	26
2.6	End-to-End Credit Management .....	27
2.7	Virtual Networks .....	27
2.8	Security .....	27
2.8.1	Firewalls .....	27
2.8.2	ATU Security .....	28
2.8.3	Configuration NOC Security .....	29
2.8.4	ARM TrustZone support .....	29
2.8.5	Suspicious Behavior .....	30
2.8.6	Encryption .....	30
2.8.7	Obfuscation .....	30
<b>Transaction Ordering 31</b>		
3.1	External Protocol Transaction IDs .....	31
3.1.1	AXI .....	31
3.1.2	OCP .....	31
3.1.3	UN2N .....	32
3.2	Symphony Ordering Fields .....	32
<b>Arteris Transport Packet Protocol 35</b>		
4.1	Overview .....	35
4.1.1	Physical Link Layer .....	35
4.1.2	Signaling .....	35
4.1.3	Virtual Channels .....	37
4.1.4	Ready and a Credit view .....	37
4.1.5	Data Link Layer .....	38
4.1.6	Network Layer .....	38
4.1.7	Transaction Layer .....	39
4.2	Messages .....	39
4.2.1	Request Message .....	40

4.2.2	Response Message .....	41
4.2.3	Message Type .....	42
4.2.4	MsgID .....	42
4.2.5	MsgLen .....	43
4.2.6	MsgSeq .....	43
4.2.7	MsgSec .....	43
4.2.8	MsgQoS .....	43
4.2.9	MsgAdd .....	44
4.2.10	NDP/CrdReq/Grant .....	44
4.2.11	Priority .....	44
4.2.12	TxHdr---[] .....	44
4.2.13	OrderingModel .....	45
4.2.14	OrderingId .....	46
4.2.15	ChannelID .....	46
4.2.16	UserCrtl .....	46
4.2.17	Non-data-payload .....	46
4.2.18	Data .....	46
4.2.19	Bytes Enables .....	46
4.2.20	RESP .....	46
4.3	ATP Data Link Layer Header .....	47
4.3.1	Packet Side-band signals .....	52
4.4	Credit Packet .....	52
4.4.1	CrMsgType[3:0] .....	53
4.5	Control Packet .....	53
4.5.1	MsgType .....	55
4.6	ATP Packet: .....	55
4.7	Packet PHITs .....	56
4.8	Packet Error Handling .....	58
4.9	ACK/NACK .....	59
4.10	Base Packet Definition .....	59
<b>Symphony Switch Architecture</b>		<b>61</b>
5.1	Introduction .....	61
5.2	Symphony Basic Flow-Through (FT) Switch .....	61
5.3	Symphony FT Switch Element Architecture .....	62
5.3.1	FT Switch Element Internal Architecture .....	64
5.3.2	Target Behavior for FT Switch .....	64
5.3.3	Timing diagram for PHIT propagation from Source to Target in FT Fabric .....	65
5.3.4	Null PHIT transfer in Flow-through fabric .....	66
5.3.5	Managing locked transfers .....	66
5.4	VC-Aware Flow Through Switch .....	67
5.4.1	Ready Aware VC-Aware Flow Through Switch .....	68
5.5	Soft Locking Arbiter (Flit Biasing) .....	68
5.5.1	Soft Locking Mechanics for Flow Through VC aware switches .....	69
5.5.2	Simulation Results .....	69
5.5.3	Deadlock Avoidance .....	71
5.6	Fully Buffered VC Switch .....	72
5.6.1	Input Virtual Channel Buffer Block .....	73
5.6.2	Flow Control Rules .....	75
5.6.3	Flow Through and VC Buffer Switch Flow Control .....	76
5.6.4	Routing .....	76

5.6.5	VC Status Fields .....	77
5.6.6	VC Allocation .....	77
5.6.7	Switch Allocation .....	80
5.7	Switch Traversal .....	83
5.8	Soft Locking Buffered VC Switch .....	83
5.9	Switch Designs Analysis .....	84
5.9.1	Input Buffered Switch .....	84
5.9.2	Input-Output Buffered Switch .....	84
5.9.3	Virtual Output Queue Switch .....	85
5.10	Starvation Protection .....	87
5.10.1	Starvation Counters in VC-Aware FT switches .....	88
5.10.2	Starvation Counters for VC buffered and Pipelined Switch .....	88
5.10.3	Cascade of Switches .....	89
5.11	Pressure Signaling .....	90
<b>Arteris Translation Unit (ATU) 93</b>		
6.1	Introduction .....	93
6.2	Overall System Architecture .....	93
6.3	Arteris Translation Unit (ATU) Overview .....	93
6.4	Supported Protocols .....	94
6.5	ATU Processing Layers .....	94
6.5.1	Native Layer .....	95
6.5.2	Common Translation Layer (CTL) .....	96
6.5.3	Symphony Message Interface .....	97
6.6	Arteris Transport (Packet) Layer (ATL/PL) .....	98
6.7	ATU Types .....	98
6.7.1	Single Ported ATU .....	98
6.7.2	Multi-ported ATU .....	99
6.7.3	Multi-ported ATP ATU .....	100
6.7.4	Streaming ATU .....	101
6.8	Custom Native Layer ATU .....	102
6.9	ATU Configuration .....	103
6.10	AXI4 Initiator ATU (max config) .....	103
6.11	AXI4 Initiator ATU (min config) .....	104
6.12	AXI4 Target ATU (max config) .....	105
6.13	AXI4 Target ATU (min config) .....	106
6.14	AXI4 ATU .....	106
6.14.1	Native Layer for AXI4 .....	106
6.14.2	AXI4 Slave Interface .....	106
6.14.3	CTL message interface .....	107
6.14.4	AXI4 Native Layer for the Target ATU .....	107
6.14.5	Common Translation Layer (CTL) .....	107
6.15	CTL Initiator block .....	107
6.15.1	Address Lookup .....	109
6.15.2	Queue mapping - .....	110
6.15.3	Input Queues - .....	111
6.15.4	Credit Manager (optional) - .....	111
6.15.5	Rate Limiter (optional) - .....	111
6.15.6	Arbiter (optional) - .....	112
6.15.7	Trace and Debug (optional) - .....	112
6.15.8	Response flow .....	112
6.16	CTL Target block .....	112

6.16.1	Queue mapping (optional) .....	113
6.16.2	Output Queues (optional) .....	113
6.16.3	Credit and Token manager (optional) - .....	113
6.16.4	Arbiter (optional) - .....	114
6.16.5	Trace and Debug (optional) .....	114
6.16.6	Response Flow .....	114
6.17	ATP/Packet Layer (PL) .....	116
6.17.1	Packetizer .....	116
6.17.2	De-Packetizer .....	117
6.17.3	Packet Layer (PL) Interface .....	117
6.18	Splitting .....	117
6.19	Exclusive Transactions: .....	118
6.19.1	Symphony Monitor States: .....	118
6.19.2	Symphony Monitor behavior: .....	118
6.20	Atomicity: .....	122
6.21	Locked Transactions: .....	122
6.21.1	Error conditions: .....	122
6.22	Non-Modifiable Transactions: .....	123
6.23	Buffered Writes: .....	123
6.24	AXI-AHB Bridge Function .....	124
6.24.1	1K Burst Crossing: .....	124
6.24.2	Protection and Memory Type .....	124
6.24.3	Address Mapping .....	125
6.24.4	Data Width Adaption .....	125
6.24.5	Timeout .....	126
6.24.6	Burst Mapping .....	126
6.25	AXI-APB Bridge Function .....	126
6.25.1	Burst Crossing .....	126
6.25.2	Protection .....	126
6.25.3	Address Mapping .....	127
6.25.4	Burst Mapping .....	127
6.26	OCP Handling .....	127
6.26.1	OCP Ordering .....	127
6.26.2	Thread Management: .....	128
6.27	ARM Kite Processor Interface: .....	128
6.27.1	Signal Integrity Protection .....	128
6.27.2	Interconnect Protection .....	130
6.27.3	Virtual Machine ID (VMID) .....	130
6.28	AXI ID, QoS and Transaction Ordering: .....	130
6.29	AMBA AXI 51 Support: .....	131
6.29.1	Atomic Transactions .....	131
6.29.2	AXI5 QoS Accept Signals .....	134
6.30	Error Handling: .....	134
<b>Memory Map</b>	<b>137</b>	
7.1	Global Address Map .....	137
7.2	Partial Address Map .....	140
7.3	Error Conditions: .....	142
<b>Adapters</b>	<b>143</b>	
8.1	Adapter .....	143
8.1.1	Adapter Functions .....	143
8.2	Bidirectional Adapter .....	143

8.2.1	Bidirectional Adapter Functions .....	144
8.2.2	Between Fabrics .....	144
8.2.3	Error Reporting .....	145
<b>ReOrder Buffer</b>	<b>147</b>	
<b>End-to-End Credit Management</b> 149		
10.1	Features .....	150
10.2	Credit Management Scheme .....	150
10.2.1	Types of Credits .....	151
10.2.2	Types Of Credit Domains .....	151
10.2.3	Theory Of Operation: .....	151
10.2.4	Credit Message .....	152
10.2.5	crdReq/Grant[9:0] .....	153
10.3	Credit Management Protocol .....	154
10.3.1	Credit Pre-allocation .....	156
10.3.2	Credit Re-sync .....	157
10.4	Token Generator (TG): .....	157
10.5	Credit Manager .....	159
10.5.1	Types of Initiators .....	159
10.5.2	Initiator ATU Credit Manager .....	160
10.5.3	CT Layer .....	160
10.5.4	Target ATU Credit Manager .....	161
10.6	Credit state and algorithms .....	162
10.6.1	Credit state variables .....	162
10.6.2	Credit Generation .....	162
10.6.3	Credit Return Algorithm .....	163
10.7	Credit Mechanism Example .....	163
10.8	Credit Domains and Virtual Channels .....	164
10.9	Credit Buffer Sizing .....	164
10.10	Credit Timeout .....	165
10.11	Network Latency and RTT Calculation .....	165
10.12	ACM Enhancement: .....	166
10.13	Hierarchical Credit Management .....	166
10.14	Credit Management and Memory Controller Interaction .....	168
10.15	Performance Results .....	169
<b>Scheduling Algorithms</b>	<b>175</b>	
11.1	Architectural Purpose and Requirements .....	175
11.2	Traffic Classes: .....	175
11.3	Scheduling Algorithms: .....	176
11.3.1	Eligibility Condition: .....	176
11.3.2	Round Robin: .....	176
11.3.3	Priority Scheduling: .....	176
11.3.4	Weighted Round Robin (WRR): .....	177
11.3.5	Modified Work Conserving Deficit Round Robin (MWC_DRD): .....	180
11.3.6	Modified Non-Work Conserving Deficit Round Robin (NWC-MDDR): .....	182
11.3.7	None-Work Conserving Weighted Round Robin (NW-WRR): .....	183
11.3.8	Weighted Fair Queueing: .....	183
11.4	Hierarchical Scheduling Algorithm: .....	184
11.4.1	Priority Round Robin .....	184
11.4.2	Hierarchical Round Robin: .....	187
11.4.3	Hierarchical Weighted Round Robin: .....	188
11.4.4	Hierarchical Priority Weighted Round Robin: .....	191

## Traffic Classes 193

### Quality of Service 195

13.1	QoS System Overview .....	195
13.2	Topology .....	195
13.3	Flow Mapping .....	196
13.4	Routing: .....	197
13.5	Transient Congestion Controls .....	197
13.6	Sustained Congestion Controls .....	198
13.7	Virtual Channels (VC) .....	198
13.8	Flow Mapping Revisited .....	198
13.9	Examples .....	200
13.9.1	Transient Congestion Controls: .....	200
13.9.2	Sustained Congestion Controls .....	202
13.10	Topology Description and Flow Mapping .....	203
13.11	Summary .....	203
13.12	Performance Measurements: .....	204
13.12.1	Latency Measurement: .....	204
13.12.2	Throughput Measurement: .....	205
13.12.3	Product Specific Measurements .....	206
13.12.4	Sampling .....	206
13.12.5	Traffic Generation .....	207
13.12.6	Network Setup .....	207
13.13	Flow-to-VC Mapping .....	208
13.13.1	ADM, ARTG and User View: .....	212
13.13.2	TCL Commands and Rules .....	212
13.13.3	GUI Representation of VCs .....	213
13.14	Effective Bandwidth .....	213
<b>Virtual Networks in Symphony 215</b>		
14.1	Symphony VN Definition .....	215
14.2	VN Requirements .....	215
14.3	Symphony Architecture for VNs .....	216
14.4	Virtual Channels (VCs) as Resources for Creating VNs .....	217
14.5	Symphony QVN Support .....	220
14.5.1	High Performance Mode .....	220
14.5.2	Pass-through Mode .....	221
14.5.3	QVN Example Comparing HP to Pass-through Modes .....	222
14.6	ARM QVN Impact on External Interfaces .....	223
14.7	ATUs and VNs .....	225
14.7.1	Initiator ATU .....	225
14.7.2	Initiator ATUs, VNs, and QoS .....	226
14.7.3	Initiator ATU and VN Processing Flow .....	226
14.7.4	Target ATU and VN Processing .....	227
14.8	VN Examples .....	228
14.8.1	Memory Controller Example .....	229
14.9	Virtual Network Hard Partitioning .....	230
<b>Configuration 233</b>		
15.1	What's considered configuration .....	233
15.2	Configuration Fabric .....	233
15.3	Configuration Register Addresses .....	234
15.4	Configuration Register Address Spaces .....	234
15.5	Configuration Adapter .....	235
15.6	Target Standard Interface .....	237
15.7	Access control .....	237

15.8	Reads and writes to non-existent registers .....	237
15.9	Configuration Target IDs .....	238
15.10	Implementation .....	238
	15.10.1 Configuration Target Interface .....	238
	15.10.2 Target Grouping .....	238
	15.10.3 Address Map .....	238
	15.10.4 Minimum Target Address Space .....	238
	15.10.5 Configuration Register Parameter Definition .....	238
<b>Errors and Interrupts</b> 239		
16.1	Interrupts .....	239
16.2	Interrupt Registers .....	240
	16.2.1 Interrupt registers at source. ....	240
	16.2.2 Interrupt Accumulation Registers. ....	241
	16.2.3 Domain Interrupt Masking Register(s) .....	242
16.3	Packetization .....	242
16.4	Errors .....	242
16.5	Implementation .....	243
	16.5.1 Initial Interrupt Hierarchy (Software starting point) .....	244
<b>Resiliency</b> 245		
17.1	Data Modifier Blocks .....	246
17.2	Pass Through Blocks .....	246
17.3	Control Blocks .....	246
17.4	Data Path Blocks .....	246
17.5	Memories and Registers .....	246
17.6	Block Composition .....	247
17.7	Block Duplication .....	247
17.8	Packet Protection .....	249
17.9	Error detection .....	250
17.10	Fault Tolerance .....	250
17.11	Safety Controller.....	250
17.12	Fail Operational .....	251
17.13	Request and Response Timeouts .....	252
17.14	Protocol Violations .....	252
17.15	Parity of ECC error detections .....	252
17.16	Spurious Packets .....	252
17.17	End to End Protection .....	252
17.18	Implementation .....	253
	17.18.1 Packet Internal Protections .....	253
	17.18.2 Phase Shift of Duplicated Logic .....	253
<b>Power and Clock Management</b> 255		
18.1	Regions vs. Domains and Sub Domains .....	255
18.2	Power & Clock Domains .....	255
	18.2.1 Domain Spanning rules .....	255
18.3	Power Management System View .....	256
	18.3.1 Power Dependency Tree (PDT) .....	257
18.4	PDT and Maestro .....	258
18.5	PDT and Power Adapters .....	260
18.6	PDT and NCORE .....	261
18.7	Configuration Network .....	263
18.8	Symphony and NCORE Solution Summary: .....	263
18.9	Examples: .....	264
	18.9.1 Example 1 .....	264
	18.9.2 Example 2: .....	268
	18.9.3 Error Management: .....	270

18.10	Interrupt and Configuration Network .....	271
18.11	Clock Gating .....	271
18.11.1	Internal Clock gating rules .....	271
18.12	Clock Interface .....	271
18.13	Power and Clock Management Domain Interface .....	272
18.13.1	Interface Rules .....	272
18.13.2	Interface definition for Clock Domains (Slave view) .....	272
18.13.3	Interface definition for Power Domains (Slave view) .....	274
18.13.4	Example Domain .....	276
18.14	Power Management States .....	278
18.14.1	Off State Behavior .....	280
18.14.2	Retention .....	280
18.14.3	Auto Wakeup .....	280
18.14.4	Abort .....	280
18.14.5	Auto Configuration on Wakeup from Power Off .....	280
18.15	Sequences .....	280
18.15.1	Active to Sleep Ready Sequence Clock Domains .....	281
18.15.2	Active to Sleep Ready Sequence Power Domains .....	281
18.15.3	Active to Sleep Ready Abort Sequence, internally aborted Clock Domains ..	281
18.15.4	Active to Sleep Ready Abort Sequence, externally aborted Clock Domains ..	282
18.15.5	Sleep Ready to Active Sequence Clock Domains .....	282
18.15.6	Sleep Ready to Active Sequence Power Domains state not retained .....	283
18.15.7	Sleep Ready to Active Sequence Power Domains state retained .....	284
18.15.8	PMA Reset to Active State Sequence .....	284
18.15.9	PMA Reset to Sleep Ready State Sequence .....	285
18.15.10	Auto Wakeup Sequence Power and Clock Domains .....	285
18.15.11	Sequence Duration .....	285
18.16	Dynamic Voltage and frequency Scaling .....	285
18.17	Future Work .....	286
<b>Performance Monitoring (PMON) 287</b>		
19.1	Definition of Performance Monitoring .....	287
19.2	PMON Components Enable PMON Functionality .....	287
19.3	PMON is Optional .....	289
19.4	Example Events .....	289
19.5	PMON interface to Trace/Debug .....	290
19.6	PMON Interrupt Capability .....	290
19.7	PMON Access .....	290
<b>Debug and Trace 293</b>		
20.1	Definition of Debug .....	293
20.2	Symphony Debug Elements .....	294
20.3	External trace interfaces .....	296
20.4	Symphony will not Support Large Explicit Internal Trace Buffers .....	296
20.5	Triggers .....	296
20.6	Trace Bus Formation .....	297
20.7	External Timestamp .....	297
20.8	External trigger outputs .....	297
20.9	External trigger or events inputs .....	297
20.10	CoreSight Compatibility .....	298
20.11	Configuration .....	298
<b>Routing 299</b>		
21.1	Objective .....	299
21.2	Oblivious Source Routing .....	299
21.3	Incremental Routing .....	300

21.4	Dynamic Routing .....	300
21.5	Routing In Symphony .....	300
21.5.1	Mapping of Flows and Virtual Paths .....	301
21.5.2	Flow Map .....	303
21.5.3	Virtual Path Map .....	304
21.5.4	Example .....	305
21.6	Deadlocks .....	306
21.7	Resource Dependency Graph (RDG) .....	306
21.8	Networks and Deadlocks .....	307
21.9	Deadlock Avoidance Algorithm: .....	308
21.10	Deadlock Example .....	309
21.11	Detecting System Level Deadlocks .....	311
21.12	Expanding RDG to include other shared resources .....	312
21.12.1	Rules .....	314
21.13	Routing Algorithms .....	314
<b>Topology Description and Mapping 315</b>		
22.1	Topology Description .....	315
22.1.1	Customer IP, Port and SoC Residual Space Capture .....	318
22.1.2	Defining Hierarchies .....	318
22.1.3	Example .....	318
22.2	Network Element Attributes .....	319
22.3	Rules and Restrictions .....	319
22.3.1	Packet Format: .....	320
22.3.2	Packetizer and Depacketizer .....	320
22.3.3	Link Format .....	321
22.3.4	Switch .....	322
22.3.5	Power and Clock Adapter .....	322
22.3.6	Width Adapter: .....	323
<b>System Analysis 325</b>		
23.1	VC Aware Flow Through Switches .....	325
23.1.1	Simulation Results .....	326
23.1.2	Solutions: .....	330
23.1.3	Case of the Ring Network .....	335
23.2	Conclusion .....	335
<b>Multicast 337</b>		
24.1	Multicast .....	337
24.2	Multicast Routing .....	338
24.2.1	Tree based Forward Multicast Routing .....	338
24.3	Deadlock Avoidance .....	341
24.4	Multicast Label and Routing Bits .....	341
24.5	Partial Address Map .....	342
24.6	Initiator ATU Multicast Support .....	344
24.7	Target ATU Multicast Support .....	345
24.8	Multicast support in the Fabric .....	345
24.8.1	Standalone FanOut Multicast Adapter .....	345
24.8.2	Multicast FanIn Adapter .....	348
24.9	Combining FanOut and FanIn Adapter - Multicast Adapter .....	350
24.10	System Level Considerations .....	351
24.11	Error Management .....	351
24.11.1	FanOut error: .....	351
24.11.2	FanIn Error: .....	351
24.12	ARTG and ADM .....	352

24.12.1 Configuring Network Elements for Multicast .....	352
--	-----

## 1

# Terminology

**Table 1-1 Terminology table.**

Term	Definition
Protocol Agent	A Protocol Agent is a source or sink of transactions and is external to the interconnect. The Protocol Agent connects to the Interconnect via an interface that uses a “Native Protocol” that is distinct than the one being described herein. The Agent can be coherent and/or non-coherent.
ATU	Arteris Translation Unit is a Symphony hardware block that connects to the Protocol Agent via its “Native Interface”.
Transaction	Defines a unit of information transfer between the Protocol Agent and the ATU.
Native Layer	Native layer is the sub-block in the ATU that interfaces with the Protocol Agent, understands the semantics of the Native Protocol and processes the transaction.
Transport Layer	A layer which provides end to end flow control for the transfer of information between an initiator and a target.
Message	Message is the granule of information that is transferred between the Native Layer and Transport Layer. A transaction is executed via one or more messages. A message can result in one or more packets at the ATP layer.
Packet	Packet contains the message and routing information that enables the Fabric to transfer it from its source to destination. A packet has a header (contains routing information) and payload (message).
FLIT	FLIT is a unit of information at which granularity the flow of information between two entities is controlled. Amount of information represented by a FLIT is based on the context of the pair of entities involved in the transfer.

**Table 1-1 Terminology table.**

<b>Term</b>	<b>Definition</b>
PHIT	A fixed amount of information that is transferred from an initiator to a target across a physical interface. A FLIT may be made up of one or more PHITs. The method of transfer could be in a clock for a synchronous interface, by a hand shake for an asynchronous interface or with a serial packet for a serial interface.
Link	A Link is a unidirectional, point-to-point means of communication between a pair of elements in an Interconnect. It is also referred to as a “Channel.” A channel comprises of two unidirectional sets of wires (one set in each direction), one carrying messages and the other carrying information controlling flow of those messages (Link Level Flow Control).
Fabric	A Fabric is set of interconnected communication elements (e.g switches, links, adapters) that work together by operating within the data link layer (packet level).
EndPoint	An End Point is a communication element that originates or terminates the Data Link Layer and sits at the extremity of a Fabric.
Initiator	Initiator is an End Point that originates a packet within a Fabric.
Target	A Target is an End Point that terminates a packet.
Upstream	Direction towards an Initiator is considered upstream.
Downstream	Direction away from the Initiator is downstream. Elements encountered along such paths are considered downstream from the Initiator.
Link Layer Credit (LL-Credit)	LL-Credit guarantees that a FLIT will be accepted on the other side of the link. LL-Credit is hop-by-hop only.
End-to-End Credit	End-to-End Credit guarantees the packet will be accepted at the End Point where the packet is expected to terminate. In Symphony End-to-End Credit Management scheme is called “Arteris Credit Management” or ACM for short.
ATP	Arteris Transport Protocol (Packet Protocol)
Adapter	Is an entity that has one input and one output and adapts communication attributes from one side of it to the other. An Adapter is inserted into a link thereby splitting the link into two parts. Adapters can contain: Power domain crossing logic, clock crossing asynch FIFOs, width changers, routing functionality that modifies the packet header before forwarding it to the next fabric.

**Table 1-1 Terminology table.**

<b>Term</b>	<b>Definition</b>
Bidirectional Adapter	A bidirectional adapter sits on two links with the property that the responses for all the requests that flow through one link return on the other. This does not imply that one of the links is only for request and the other is only for responses.
Virtual Channel (VC)	Virtual channel is a mechanism that creates mutually independent communication channels that use the same physical channel by time division multiplexing the use of a physical channel on a per clock cycle basis for use in transferring PHITs of distinct end-to-end message communications.
VC-lite	Any network or fabric where only some of the components are VC aware.
Channel FIFO	Standalone FIFO that serves to hold PHITs. This FIFO can be inserted in a link between switches and/or ATUs.
Virtual Channel FIFO (VCFIFO)	Is a Channel FIFO that serves to hold PHITs belonging to a virtual channel.
Virtual Network (VN)	A virtual network is composed of a set of End-Points with a set of policies that govern how information is transferred between those end-points. A virtual network may be composed of multiple Fabrics. Multiple virtual networks can share the same physical network. The policies that a VN can have may include Resiliency, QoS, Fault tolerance, and trace points.
Interconnect	An Interconnect is composition of one or more Fabrics, connected through Adapters.
Response Re-order Buffer	Buffers required to store responses that require re-ordering to meet ordering requirements of the receiving protocol.
Switch	A switch is a communication element with one or more ingress/egress ports and which enables the packets to be transferred from an ingress port to an egress port on a PHIT basis.
Data path	The path taken by the PHITs across the switch is called data path.
Control path	The path taken by control information across the switch for coordinating the transfer of the PHITs is control path.
Flow Through Switch (FT-Switch)	FT-switch has only combinational logic in the data path. The control path, on the other hand, has sequential elements.

**Table 1-1 Terminology table.**

<b>Term</b>	<b>Definition</b>
Pipelined Switch (PL-Switch)	PL-switch has sequential elements in both control and data path.
System Address Map (SAM)	It is a mechanism to associate targets with portions of accessible address space within the system. It helps in translating a transaction's address into a network identifier of the target device.
Partial Address Map (PAM)	Partial address map contains a subset of the addressable space of SAM and may provide translation for a subset of targets in the system.
Domain	Domain is a grouping of elements in the Interconnect that share one or more defining characteristic. When used in the same context as a region, will encompass a smaller group than a region. For example, a clock region can contain multiple clock domains, but not the other way around.
Power Domain	Grouping of elements in the Interconnect that share the power supply and a single control interface driven from the PMU.
Clock Domain	Grouping of elements in the Interconnect that share the same clock and a single control interface driven from the PMU.
Region	Region is a grouping of elements in the Interconnect that share one or more defining characteristic. When used in the same context as a domain, will encompass a larger group than domains. For example, a clock region can contain multiple clock domains, but not the other way around.
Power Region	Grouping of elements that share a power supply. A Power Region can contain multiple Power Domains and Clock Domains
Clock Region	Grouping of elements that share a clock supply. A clock Region can contain multiple Power Domains and Clock Domains.
Credit Domain	Logical grouping of elements in the Fabric that support end-to-end credit management protocol.
Stream	A Stream is a set of messages that are mapped to a single fabric and is uniquely defined by the tuple {SrcID, DestID, QoS, VN, address}. Streams are unidirectional.
Flow	A flow is end-to-end and can be composed of multiple Streams. Flows are unidirectional. A flow can have multiple attributes associate with it. The Flows are defined to be between End Points of the Interconnect.

**Table 1-1 Terminology table.**

<b>Term</b>	<b>Definition</b>
Path	A path is a distinct sequence of Interconnect components (ATUs, switches, links and adapters) and has the following properties: 1) Has a beginning and an end; 2) Each component in the sequence is adjacent to another; 3) No link is repeated in the path; and 4) path is unidirectional.
Virtual Path	A virtual path is the same as a Path, but is also assigned to virtual channels. Multiple Virtual paths may be mapped on a single link.
PMU	Power Management Unit. A block that exists outside the interconnect and drives the control interfaces to clock and power domains.



# 2

# Introduction

This document defines a scalable interconnect architecture and its associated components. The interconnect is intended to provide connectivity for various types of agents in an SoC including processors, peripherals, memory elements (controllers/schedulers), sophisticated hardware accelerators, hardware elements that handle packetized IOs, and other IP blocks.

The Interconnect is based on a sophisticated underlying packet transport layer and End Points. The End Points (ATU), convert native protocol in to Arteris Transport Packet Protocol (ATP).

The architecture does not dictate any topology restrictions. The architecture also comprehends the ability to traverse multiple localized instances of the fabric within a single SoC and/or possibly across multiple SoCs.

The interconnect along with its other hardware and software components is called Symphony.

## 2.1 Architectural Goals and Requirements

The Symphony architecture is designed to provide the interconnect for a wide range of systems. The focus of the current release of Symphony is the mid-to-high end systems. Later releases would encompass the high and low end systems.

Key features of the architecture are:

- ▶ Scalable architecture that can address the interconnect needs of low to high end systems.
- ▶ Non-blocking and non-interfering architecture.
- ▶ Layered architecture approach that delineates functionality at each layer.
- ▶ Designed to support multiple types of standard and custom Protocols.
- ▶ Packet based communication inside the Interconnect.
- ▶ Designed to support the following transaction types:
  - ◆ Request/Response transactions.
  - ◆ Streaming transactions.
  - ◆ Cache Coherent transactions.
- ▶ Support for end-to-end Quality of Service (QoS).
- ▶ Support for End-to-End Credit Management.
- ▶ Support for Resiliency.
- ▶ Support for fault tolerance.
- ▶ Support for multiple virtual and partitioned systems.

- ▶ Support for ARM TrustZone.
- ▶ Configurable data width.
- ▶ Support for error reporting and logging.
- ▶ Support for Performance monitoring.
- ▶ Support for Multiple Power and clock domains.
- ▶ Support for Debug and configuration management.
- ▶ Extensive Power Management support at multiple levels.
- ▶ ARM P/Q-Channel compatible support for managing power via an external PMU.
- ▶ Modularized and pipelineable architecture to support physical awareness/constraints.
- ▶ Support for Ordering/Reordering transactions
- ▶ User Access language and sandbox.

## 2.2 Layered Architecture

Symphony is based on a layered architecture. Each layer within the architecture performs a specific function.

This document will describe each of these layers and their associated functions.

These layers are:

- ▶ Transaction Layer Processing
- ▶ Network Layer Processing (optional, for traversing multiple instances of a fabric)
- ▶ Data-Link Layer Processing
- ▶ Physical Layer Processing

### 2.2.1 Transaction Layer

The Transaction layer understands the needs of the native protocol. This layer handles protocols such as AXI, APB, CHI, etc. The Transaction layer receives transactions from the Protocol Agents and performs the functions necessary to convert the transaction into packets before passing the information to the Network or the Data-link layer as necessary. At the destination end, the Transaction layer receives packets from the Network layer or the Data-link layer and converts the received packets into relevant protocol transactions. The Transaction layer then delivers the transactions to the Protocol Agents.

---

*Note: A Protocol Agent is a source or sink of transactions. The Agent can be coherent and/or non-coherent. For example, a core complex connected to the interconnect is an Agent. Similarly, a Security IP block connected to the interconnect is an Agent. We can extend the definition of the Agent to include packetized as well as non-packetized interfaces and other IP blocks in the SoC.*

---

The Transaction layer provides the services necessary to support different native protocols (CHI, AXI, OCP, etc.).

## 2.2.2 Network Layer

Network Layer contains information that is needed to determine the route of the packets from Source to the Destination in the Interconnect, that is, this layer contains the address of the location the transaction is destined to. This layer is also responsible for Traffic Management across the fabric. Below is a list of functions that the Network Layer will perform:

- ▶ Contains information needed to determine the Route from Source Node to the Destination Node.
  - ◆ Across a single or multiple Fabrics.
- ▶ Traffic Management.
  - ◆ Credit management
- ▶ Manages other features such as support for end-to-end Security, Fault Tolerance and Reliability.

## 2.2.3 Data Link Layer (Packet Layer)

The Data link layer provides the framing, the functions and procedures required to transfer packets between the different entities of the Interconnect. These include creating the switching headers, packet/flit switching, contention resolution at the packet/flit level, etc. The Data link layer is also responsible for link-by-link flow control and hop-by-hop credit management.

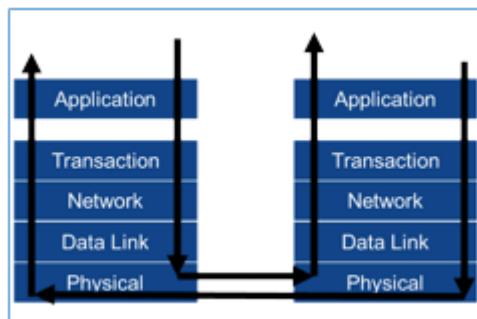
Below is list of functions that this layer will perform:

- ▶ Moving packets hop-by-hop across the Fabric
- ▶ Packet Switching – Assigning inputs to output
- ▶ Transient congestion resolution: Priority, Fair bandwidth allocation, starvation protection.
- ▶ Link-by-link flow control (Link-by-Link credit management)

## 2.2.4 Physical Layer

The Physical layer is concerned with the format/framing of packets/flits and the actual transmission of control and data bits over the physical channels.

**Figure 2-1 Symphony Layered Architecture**

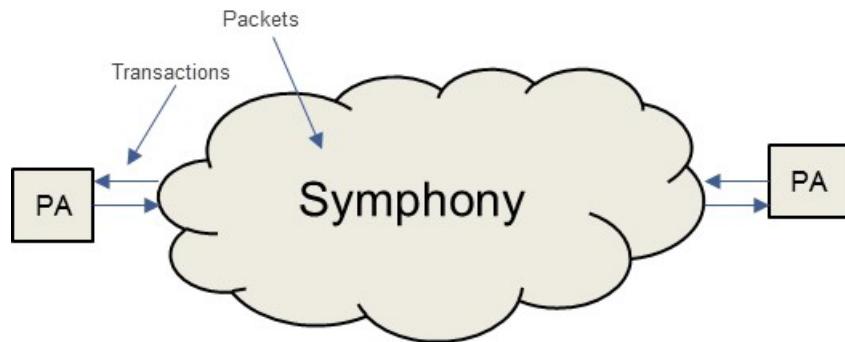


## 2.3 Overall System Architecture

The Symphony system acts as the underlying transport for moving transactions between different Protocol Agents. Symphony uses packets to carry data and control information between the source and destination.

Multiple instances of the fabric are also possible, connected to one another via units called Adapters (to be described later in the document).

**Figure 2-2 Symphony Overall System Architecture**



At the periphery, Symphony connects to the native Protocol Agents. It converts native protocol transactions into packets and transports them across the Fabric. At the target, it converts these packets back into native protocol transactions.

Fig 1.

SW

## 2.4 Topology

The Symphony architecture is topology agnostic and will enable any topology needed to connect the relevant components on the SoC.

Symphony will support both irregular as well as regular topologies. There will be a segment of our customers who will require irregular topologies especially in the low to mid-tier designs.

Note: Recently our customers have started to ask for regular topologies as well.

A regular network topology is defined in terms of a regular graph structure (such as a mesh, ring, torus, etc.). In a regular topology, each node has the same numbers of neighbors, and hence regular topologies are extensible by simply adding stages to the previous structure. Since regular topologies are based on a regular graph structure, routing is relatively easier in such structures.

Irregular topologies do not have a regular graph structure. The Symphony architecture will support Endpoint interconnect architecture referred to in the literature as an indirect Interconnect. However, the architecture will not preclude a direct interconnect structure as well.

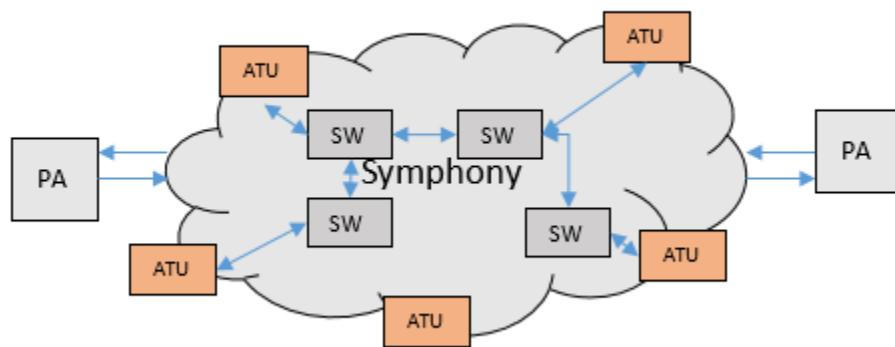
Topologies such as Torus and Mesh lend themselves naturally to a direct<sup>1</sup> interconnect structure, while others such as Benes multistage networks, ShuffleNets etc. are more affine to indirect/Endpoint interconnects.

The topology of the Interconnect is a critical design decision for the SoC, as it directly impacts the zero load latency and the overall throughput of the Interconnect.

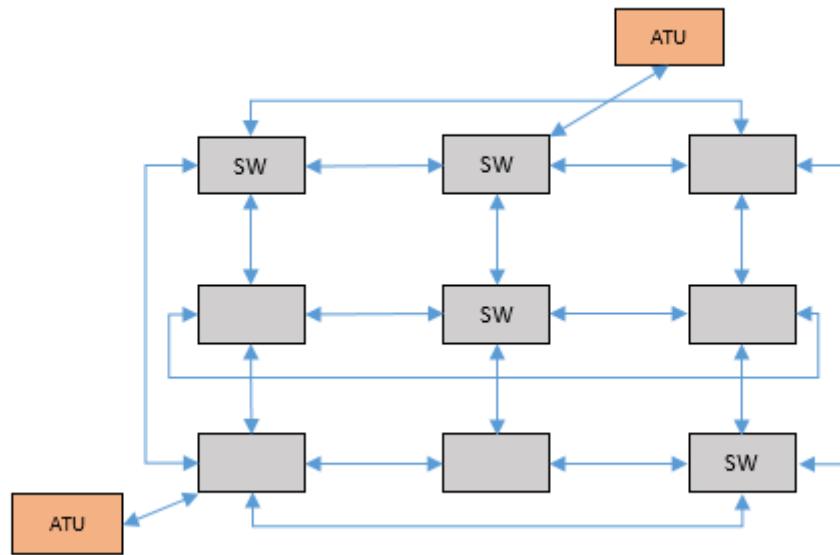
An irregular endpoint interconnect structure and a regular Torus direct interconnect are shown in Figure 2-3 on page 25 and Figure 2-4 on page 26, respectively. A Ring topology is simply a 1-D Torus Interconnect.

Note that, to support a 2D Torus or a Mesh Direct Interconnect, the switches would have to support 5x5 and/or 9x9 configuration respectively. For a 3-D Torus the switches would have to support a 7x7 configuration.

**Figure 2-3 EndPoint Irregular Topology**



1. Direct interconnects are those for which the endpoints sit inside the Interconnect. Indirect interconnects are those for which the endpoints sit outside the Interconnect.

**Figure 2-4 2D Torus Network**

## 2.5 Quality of Service (QoS)

The Symphony system will offer different classes of service ranging from best effort to deterministic latency and bandwidth guarantees.

The QoS mechanism in Symphony will enable the system to offer the following:

- ▶ Predictable latency through the Interconnect.
- ▶ Minimum and maximum bandwidth guarantees.
- ▶ Priority mechanism for latency sensitive traffic.
- ▶ Starvation protection for lower priority traffic.
- ▶ Fair bandwidth allocation where required.
- ▶ Best effort traffic support.

The above listed features/capabilities are achieved by supporting different QoS mechanisms such as:

- ▶ Priority and fair bandwidth scheduling policies at the ATUs and Switches.
- ▶ End-to-End and Link-by-link credit flow control to ensure high switch and link utilization.
- ▶ Buffer management at the ATU.
- ▶ Virtual Channel support where required in the Interconnect.

## 2.6 End-to-End Credit Management

The QoS features described in the previous section assume a constant drain rate at the destination node, however, that may or may not be the case. The drain rate may fluctuate based on the conditions at the downstream Protocol Agent. To offer predictable QoS through the system, a feedback loop has to be provided from the destination to the source to adjust the injection rate of traffic to match the conditions at the destination.

End-to-end credit management offers such a feedback mechanism.

Symphony has a robust end-to-end credit management protocol that will allow the system to recover from lost/corrupted credits.

The Credit Management system architecturally allows for multiple credit domains to co-exist in the network. A Credit Domain is defined as collection of ATUs that participate in a particular Credit Management scheme. A single ATU can be a member of multiple credit domains.

## 2.7 Virtual Networks

Virtualization abstracts the underlying physical hardware and separates the functionality from the physical implementation of the hardware. It is a combination of software and hardware that together makes the underlying physical resources appear as autonomous logical environments or systems.

A Virtual Network is an abstraction of the physical SoC (System on Chip), complete with its view of the interfaces, memory and other resources within the physical SoC. Each virtual network is functional and completely segregated and has the ability to operate and appear as a physical device. For example, consider a multi-tenant router (a router that handles traffic from multiple businesses and customers) where each tenant would like to receive the SLA (service level agreement) that they have paid for. This SLA may include uptime, bandwidth, privacy, and/or security guarantees. Traditionally routers would support multi-tenancy by using discrete SoCs. However, with the wide adoption of Virtualization it is now possible for the router to use a single SoC and offer multi-tenancy with SLAs similar to that offered by discrete devices.

Symphony support for Virtual Networks is described in detail later in the document.

## 2.8 Security

### 2.8.1 Firewalls

Firewalls exist on links and have access to info in the packet to make decisions about security associated with the packet.

#### 2.8.1.1 When security is violated

---

Firewall will mark a packet as bad and pass it through. Target ATUs will behave appropriately when receiving such a packet.

### **2.8.1.2Customer Custom Security Code**

The customer will have the ability to create custom RTL to implement security features in the code. An example of this would be read and write restrictions based on address ranges or other customer visible information that exists in a packet.

### **2.8.1.3Customer Controllable Packet Bits**

A field will be provided and method of writing and reading those bits will be provided. Customer Custom Security Code will have access to the bits.

### **2.8.1.4Configuration Registers**

The firewall in the NOC can be configurable based on registers and the Customer Custom Security Code will have access to those configuration registers.

---

Implementation  
This means that there will be registers that a customer can configure the format of, those registers can be read and written through a configuration interface as described in Configuration on page 233 as well as be modified or accessed by the Customer Custom Security Code.

---

### **2.8.1.5Customer Visible Side Band signals**

The firewall will have access to customer visible sideband signals that enter and exit the Symphony NoC and the Costumer Custom Security Code will have access to those signals.

### **2.8.1.6Virtualization Features**

The partitioning of a NOC for virtualization reasons will use firewalls. These firewalls will have access to parts of the packet that the customer will not for his custom firewall RTL.

## **2.8.2ATU Security**

### **2.8.2.1Firewalls on Links**

The majority of the security features will be associated with firewalls that sit on the links directly connected to the ATU.

## 2.8.2.2DoS

The exception to this is DoS prevention. All DoS prevention security needs to be implemented at the Requester ATU logic. As an implementation detail, this can be achieved with the use of a special firewall directly connected to the ATU. The firewall behavior will be different than standard. Instead of marking a packet as bad, the firewall will kill the packet and have a signal directly back to the ATU indicating that it has done so.

## 2.8.3 Configuration NOC Security

Configuration Security will be achieved through the use of firewalls in the configuration NOC in a similar manner that is done in the request/response NOCs.

### 2.8.3.1 Physical Separation of Configuration NOCs

In the same way that request types can be separated between ports on an ATU, the same can be done in the configuration NOC to create physically separate configuration networks.

## 2.8.4 ARM TrustZone support

ARM TrustZone bit will be in the packet and accessible by firewalls and ATUs. ATUs will support two modes as described below.

### 2.8.4.1 Mode One

ATUs Pass TrustZone bit through. It is up to customer to implement TrustZone features.

Firewalls have access to TrustZone bit.

### 2.8.4.2 Mode Two

Firewalls have access to TrustZone bit.

### 2.8.4.3 ATU Masters

Pass TrustZone bit through.

### 2.8.4.4 ATU Targets

There will be a register in the ATU that hangs off the ATU address space and is always in secure access mode. This follows the ARM model. This will be referred to as the ATU TrustZone Register for specification purposes.

---

Bit per target: Multiple Targets can be hanging off an ATU. There will be a bit per target in the ATU TrustZone Register that states what TrustZone security state the Target is in.

ATU behavior: Based on the request type and the target's TrustZone security state, the ATU will block transactions that are going to an inappropriate target, return the appropriate error response, store state associated with the violation and assert an interrupt.

There will be a method provided to make a particular target bit in the ATU TrustZone Register hard coded and not programmable.

There will be a method provided to state the default values for programmable bits in the ATU TrustZone Register.

## **2.8.5Suspicious Behavior**

The performance monitors can be configured to count specified events and generate an interrupt, when the count exceeds or not exceeds a pre-configured count. Software can take appropriate action when it services the interrupt.

The control and configuration of these performance monitors can be protected through the use of fire-walls and the use of features documented in the configuration section of this document.

## **2.8.6Encryption**

Encryption is outside the scope of the Symphony Interconnect.

## **2.8.7Obfuscation**

At present, there are no plans to obfuscate data and packet contents. However, because of the configurable nature of Symphony an amount of obfuscation happens naturally. Packet contents and data paths vary as a function of configuration. Separately configured NOCs on the same chip can have very different packet contents and data path organization.

# 3

# Transaction Ordering

Transaction Ordering rules are enforced at the ATUs, Adapters and Reorder buffers. Packets are marked with the ordering rule class they must follow and an ordering ID.

---

*There is the potential to dramatically increase the performance of PCIE style order rules by embedding enforcement along the links. In all these implementations responses for requests must flow through the same hardware element, which is the fundamental rule for the placement of a bidirectional adapter. Therefore, these enhancement can be implemented in the context of this specification through the use of adapters.*

---

## 3.1 External Protocol Transaction IDs

Generically, If the external protocol uses transaction IDs as a part of its ordering protocol, Symphony will order the transactions based on these IDs regardless of other contexts unless the external protocol explicitly allows or demands it.

### 3.1.1 AXI

Transactions with the same AXI IDs and same type of transaction from the same Initiator will be ordered. That is, if the transactions have the same target ID, then they will be arrive in the same order at the Target as they left at the Initiator. If they have different targets, the responses sent to the Master by the Initiator ATU will be in the same order as the order of the transactions received.

No ordering is maintained between reads and writes. If ordering is required between reads and writes then the Master should ensure that the earlier transactions is complete before it issues the next transaction.

---

————— Note ————  
See section A5.3.4 AMBA5 ARM IHI 0022f.b (ID 122117).

---

AXI protocol gives precedence to AXI ordering rules over QoS. Symphony will honor the same.

### 3.1.2 OCP

OCP rules state that ordering is limited to transactions within the same thread and having the same Tag ID. It does not matter whether the transaction is read or write. There are no ordering requirements for transactions belonging to different threads.

Transactions with overlapping addresses within the same Thread whether or not they have the same Tag ID are committed in order. The responses can be reordered, but responses with the same Tag ID should be returned in order as well.

### 3.1.3UN2N

The UN2N protocol ordering model is similar to OCP ordering model. Transactions with the same Ordering ID within the same Channel ID have to be ordered with respect to one another, but not if they belong to different Channel ID. All overlapping address even with different Ordering IDs have to be ordered with respect to one another if they belong to the same Channel ID. This ordering model is similar to the OCP ordering model.

## 3.2 Symphony Ordering Fields

Symphony packet protocol and the CTL layer interface define the TxHdr field. The TxHdr field is expanded to include the ordering ID and ordering model fields. These fields are defined as follows:

The ordering fields are defined as follows:

- ▶ TxHdrOrderingID = {OrderingID}
  - a. For AXI the mapping is: TxHdrOrderingID = {AXI ID}
  - b. For UN2N the mapping is: TxHdrOrderingID = {OrderingID}
  - c. For OCP the mapping is: TxHdrOrderingID = {TagID}
- ▶ TxHdrChannel = {Channel/Thread/TrafficClass/Type of Traffic}
  - a. For AXI the mapping is: TxHdrChannel = Write/Read Transaction
  - b. For UN2N the mapping is: TxHdrChannel = Channel ID
  - c. For OCP the mapping is: TxHdrChannel = Thread
  - d. For PCIe the mapping is: TxHdrChannel = Traffic Class (TC)
- ▶ TxHdrOrderingModel = {AXI/OCP/UN2N/PCIe/Others}

**Table 3-1 Ordering model field**

Protocol	TxHdrOrderingModel[3:0]
AXI	[0001]
UN2N	[0010]
OCP	[0011]
PCIe	[0100]
Ready Transactions only	[0101]
Write Transactions only	[0110]
Reserved	[0111]-[1111]

The TxHdrOrderingID field establishes the primary ordering relationship between transactions. Additional ordering relationship is established by the TxHdrChannel field. For example, there are additional ordering requirements between transactions belonging to the same channel such as ordering between overlapping addresses even if the Ordering ID is different.

Transactions are ordered based on the following fields:

- ▶ {TxHdrOrderingID, SrcId, SeqNum, TxHdrChannel} and {TxHdrOrderingModel}

The ordering model field provides additional information on how the transaction needs to be treated. For example, Symphony will order both read and write transactions with the same Ordering ID if they belong to the same Channel. Similarly, only writes with respect to other writes would be ordered for AXI ordering model.



# 4

# Arteris Transport Packet Protocol

## 4.1 Overview

ATP is a layered protocol and is composed of the following:

- ▶ Data link layer (Packet layer)
- ▶ Network layer
- ▶ Transaction layer

### 4.1.1 Physical Link Layer

The physical layer contains the definition of the signaling and protocol that exists at the physical layer. There can be more than one implementation of the physical layer and as long as it is able to duplicate the functionality described then it is equivalent and can be used in place of what is specified here.

---

Implementation

The following is an implementation example but may not encompass totally the actual implementation.

---

### 4.1.2 Signaling

The physical Layer Signaling is called an ATP interface. The Table below specifies the interface from the point of view of the initiator..

**Table 4-1 Physical Layer Signaling**

Signal	I/O	Description	Width	Optional	Parameterizable
valid	O	1'b1 Indicates the present beat has valid data from master	Number of VCs	No	Yes
ready	I	1'b1 Indicates slave can accept beat this cycle	Number of VCs	No	Yes
first	O	1'b1 indicates first beat of packet.	1	No	No

**Table 4-1 Physical Layer Signaling**

Signal	I/O	Description	Width	Optional	Parameterizable
last	O	1'b1 indicates last beat of packet	1	No	No
bus	O	Contains the contents of packet	variable	No	Yes
pressure	O	The pressure of the present beat	variable	Yes	Yes
prot	O	The protection bits of the outputs of the present beat	variable	Yes	Yes
prob_b	I	The protection bits of the inputs of the present beat	variable	Yes	Yes

### 4.1.2.1Protocol

A message is encoded into a packet and a packet be from 1 to many beats.

Valid=1 indicates that the other outputs have meaning for the present “beat.” A beat is a slice of time that may be broken up by the use of the clock signal. When valid is not 1, none of the other outputs have meaning and so can have any value.

Valid can be a function of the present beat ready, but this use should be restricted to very specific use cases.

---

#### Implementation

If there are two or more back to back interfaces with no pipelining in between, only the last interface can have valid be a function of ready. All other combinations will cause logic loops.

---

Ready=1 indicates that the slave can accept a beat of data for the present beat. Ready shall never be a function of valid.

First=1 means the present beat is the first beat of a packet. Last=1 means the present beat is the last beat of a packet. First and last can both be 1 on a single beat.

---

#### Implementation

The hardware that carries packets across the network should not pay attention to the size of the packet embedded in the packet, but only to the first and last signals.

---

### 4.1.2.1.1Pressure

When a fabric has a packet definition that includes priority, the pressure signal can be optionally enabled. Pressure is one hot encoding of the present pressure of the present beat of data.

Pressure has two use cases.

The first use case is pressure is the highest priority of all the packets presently existing between a fixed interface and all the initiators that can use that interface including the pressure being presented by the initiators. Pressure out of an initiator can be different than the priority of the present packet being presented. Pressure affects arbitration, but only when the interface does not have ready=1. So, if ready=1 then the priority of the present packet is used for arbitration. If ready=0, then the pressure is used on the next cycle for arbitration and is pushed forward to the next interface that the present packet.

The second use case of pressure is when starvation avoidance is enabled in priority arbiters being used by the network. When starvation avoidance kicks in, the packet that caused it will have its priority instantly raised to the highest priority in the system. In this case, this priority is always used, regardless of state of ready. When starvation is enabled, if a system has N priorities, then the pressure signal will have a width of N+1 and only packets that have triggered starvation can use the highest priority.

The two use cases can be mixed. When this happens the starvation avoidance pressure bit is sticky on the packet that caused it and it pressures forward as the next highest priority.

#### **4.1.2.1.2Prot and Prot\_b**

When protection is added to the interfaces all the protection bits will be concatenated together and put on the prot and prot\_b signals. Prot is for protection on the all the outputs and prot\_b is for protection on all of the inputs (of which there is only one.)

The protection signal encoding function can change as a function of beat of a packet.

#### **4.1.2.2Retraction**

If first is asserted with valid and ready is not asserted, then on the next beat, the present packet can be retracted and another put its place. However, if ready is asserted at the same time as first and valid, then that packet must be completed before it can be changed.

#### **4.1.3Virtual Channels**

Virtual channels (VCs) are added to the interface by vectorizing the ready and valid signals. Virtual Channel 0 corresponds to valid[0] and ready[0] and so on. The valid vector is 1 hot 0, meaning that only one virtual channel can be presenting at a time or none. The ready vector has no restrictions on it.

The protocol descriptions above are separated per Virtual Channel. Therefore, once a packet starts in a VC, you can't switch to a different packet within that VC, but the physical link can change VCs every cycle.

#### **4.1.4Ready and a Credit view**

One way to view the ready signal is it is the target telling the initiator that there is a credit available.

##### **4.1.4.1Extending Past a Single Credit**

One easy way to extend this is to change the definition of ready to indicate the return of a credit per cycle. The change in the logic is very slight.

On the initiator side instead of connecting the ready signal directly to the internal logic, have the interface ready signal increment a counter and have the internal valid decrement the counter. If the counter is greater than 0, then internal valid is passed to external valid and hold internal ready high, otherwise external valid is 0 and internal ready low.

On the target side, there is typically buffering associated with the incoming data. When the buffer gains a spot, it asserts external ready. When the buffer is not full it asserts internal ready.

On power up or initialization the easiest approach is to pre allocate the credits by initializing the counter on the initiator side to the number of spots available.

#### **4.1.4.2 Credits and Virtual Channels**

When an interface has more than one virtual channel, than there needs to be a counter per virtual channel on the initiator side.

#### **4.1.5 Data Link Layer**

The data link layer contains the fields necessary to transfer the packets between different entities in the interconnect. These fields include, among others the following:

- ▶ Information necessary to route the packets in the Interconnect.
- ▶ Quality of service associated with the packet
- ▶ Security settings
- ▶ Virtual Network association
- ▶ Packet type
- ▶ Error types

#### **4.1.6 Network Layer**

Network Layer contains the information needed to route packets in the Interconnect. This layer is also responsible for Traffic Management across the fabric. Below is a list of Network Layer functions:

- ▶ Contains information to determine the Route from the Initiator ATU to the Target ATU.
- ▶ Credit management.
- ▶ Splitting of transactions and merging of packets.

The Network Layer field contains the information required to perform the above functions and contains, among others the following:

- ▶ Source ATU ID.
- ▶ Destination ATU ID.
- ▶ Address of the transaction.
- ▶ Transaction size

Symphony defines a transaction by the following:

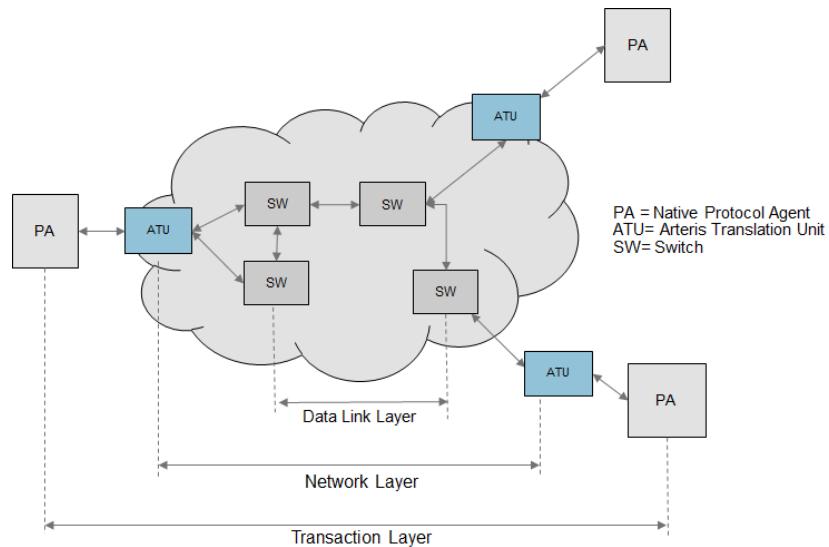
- ▶ The type of the transaction
- ▶ Transaction attributes, .e.g., bufferable, non-bufferable, posted, non posted etc.
- ▶ Transaction profile such as burst type (Increment, fixed, wrap), burst length.
- ▶ Special consideration such as type of Atomics and whether the transaction is exclusive, locked etc.

## 4.1.7 Transaction Layer

The transaction layer carries information needed to appropriately interact at the transaction layer. That is it carries information that is relevant for the intended action or function at the Native Layer.

This layered approach is illustrated in Figure 4-1 below.

**Figure 4-1 : Layered Packet Protocol Approach**



## 4.2 Messages

As was mentioned above, ATP is a layered protocol and as such is internally composed of ATP messages which are encapsulated in ATP packets.

The native transactions are converted into messages, and the messages are then mapped into packets (and vice-versa). The message fields for request and response message are shown below. Note: that there are no architectural limits for these fields. Limits on these fields would be based on the overall system as determined by the user.

## 4.2.1 Request Message

**Table 4-2 Request Message**

Fields	Description	Bit Range <sup>a</sup>	Value Range <sup>a</sup>	Parameterizable
MsgID	Transaction identifier	Depends on system configuration	[-:0]	Yes
MsgLen	Transaction size	Same as above	1B-4KB	Yes
MsgSeq	Sequence No. for multiple outstanding transactions (e.g.: using the same AXI ID)	Same as above	[-:0]	Yes
MsgSec	Security level of the transaction	2		Yes
MsgAdd	Address associated with the transaction	Same as above		Yes
MsgType	Type of Message	3		Yes
MsgTypeAttri	Attribute of the Message	2		No
MsgQoS	QoS levels associated with the transaction	0-4		Yes
MsgStatus	Defines the status of the msg	2		No
MsgOrdering-Model	Defines the ordering model this transaction belongs to	4		Yes
MsgOrderingID	Defines the ordering relationship between transactions with the same ordering ID	Depends on system configuration		Yes
MsgChannelID	Defines the relationship between transactions with the same channel ID but different Ordering ID	Depends on socket configuration		Yes
TxHdrBurstType	Defines the type of burst	2		Yes
TxHdrBurstSize	Defines the number of bytes in the beat	Depends on socket configuration		Yes

**Table 4-2 Request Message**

Fields	Description	Bit Range <sup>a</sup>	Value Range <sup>a</sup>	Parameterizable
TxHdrTrnMem-Attr	Defines the memory attribute of the message such as whether the msg is modifiable or not.	3		Yes
TxHdrRegion	Carries information about which region the transaction belongs to	Depends on how many regions are defined		Yes
TxAtomicAttr	Describe the atomic transaction attribute	3		Yes
UserCrtl	User define signals	Depends on socket configuration		Yes
NdpdData	Non-data payload data	Depends on system configuration		Yes
Data section	Data section is decoupled from rest of the message			
Data	Data width	Depends on system configuration		Yes
BE	Byte Enables	Data Width/8		Yes
Last	Last beat	1		
UserData	User defined fields	Depends on socket configuration		Yes

a. There are no architectural limits. Limits would set based on the overall system.

## 4.2.2 Response Message

The response message is similar to the request message with the difference that it does not need to carry Transaction attributes.

## 4.2.3 Message Type

In Symphony message types are divided in to two broad categories:

- ▶ Requests messages that inform the Slave or the terminating agent of an action that needs to be performed by the Slave or the terminating agent. Request messages can be accompanied by data. Generally when request messages are accompanied by data, then the action needs to be performed on the accompanying data. Request messages can also be accompanied by meta data.
- ▶ Response messages that inform the Master or the agent that generated the request of how the request was complied by. Response messages can be accompanied by data and/or meta data.

Message Type identifies the type of message. ATP will support the following types of messages (credit and control messages are described separately):

- ▶ Request
- ▶ Request with data
- ▶ Response
- ▶ Response with data
- ▶ Request with meta data (Non-payload data)
- ▶ Response with meta data (Non-payload data)
- ▶ Request with meta data and data
- ▶ Response with meta data and data

**Table 4-3 Message Types.**

Msg Types [2:0]	Message Type
[--0]	Request
[--1]	Response
[-0-]	without meta data
[-1-]	with meta data
[0--]	without data
[1--]	with data

## 4.2.4 MsgID

This field defines the identification that the Interconnect assigns to each Transaction. The MsgID should be combined with the tuple that defines the flow to uniquely identify the message. A flow can be identified by the tuple {SrcID, DestID, VNID, QoS, address}. MsgID is assigned by the CT layer or the AIUs and is delivered to the packet layer via the SMI interface.

## 4.2.5MsgLen

This field defines the length of the message in bytes (+1). MSG length may or may not be the same as the

**Table 4-4 Message Length.**

MsgLen	Size (Bytes)
0x00	1
0x01	2
---	----
0xFF	256
---	---
FFF	4096

packet length. A Msg can traverse multiple packets.

## 4.2.6MsgSeq

This fields defines the sequence number of the outstanding transaction with same AXI ID. Symphony can support up to 512 outstanding transactions in large systems and 16, 32, 64 outstanding transactions in normal systems. MsgSeq number will be used for reordering, if transactions are for example stripped to different memory controllers. MsgSeq number is also used to number packets that belong to a single message. The ATU will back pressure if it has run out of sequence numbers.

## 4.2.7MsgSec

This field defines the Security level of the transaction. This security field maps in to the security field defined in the packet header. User defined bits can also be mapped to MsgSec field.

**Table 4-5 Message Security.**

MsgSec	Definition
-0	Unprivileged
-1	Privileged
0-	Instruction
1-	Data

## 4.2.8MsgQoS

This field defines the QoS associated with the Message. Selected user defined bits can be mapped to this field.

## 4.2.9MsgAdd

This field defines the physical address associated with the message. This field is parameterizable. Symphony allows address compression if configured accordingly. MsgAdd field can also be used for hierarchical addressing in the Fabric/Interconnect.

## 4.2.10NDP/CrdReq/Grant

This is an overloaded field. In NCore this field will be used to carry the non-data payload bits in the header. For Presto, this field will be used for control messages such as Credit request/Grant, end-to-end error, etc. This field is defined in Table 4-6.

## 4.2.11Priority

This field is carried across the network.

**Table 4-6 Credit Msg Type.**

Credit/Grant/Control[4:0]	Type of Credits Requested/Granted
00xxx	NDP non-data payload
01xxx	Credit msg
10xxx-11xxx	Reserved
010x0	Credit_requested for Request/command Transactions
010x1	Credits requested for Write Data
01x1x	Dedicated Credit if set, otherwise Pool credit
011x0	Credits granted for request transactions
011x1	Credits granted for Write data

**Table 4-7 Credit Request Type Continued.**

CrdReq[12:5]	Number of Credit Requested/Granted
0x00-0xFF	0-255

## 4.2.12TxHdr---[]

This field contains the transaction layer attributes. The transaction layer attributes are contained in the different subfields of the TxHeader. These attributes are divided in to the following categories:

- ▶ Transaction profile which captures information about the burst type.

- ▶ Transaction attributes such as modifiable, bufferable attributes of the transaction.
- ▶ Special considerations such Normal/Atomic or Exclusive operations.

Table 4-8, provides details of how a transaction attribute is defined in Symphony and captured by the transport protocol.

**Table 4-8 TxHdr-- fields**

TxHdr---	Definition
TxHdrBurstType[1:0]	[00] Fixed [01] Incr [10] Wrap [11] Reserved
TxHdrBurstSize	Burst size
TxHdrTrnAttr[3:0]	Indicates if the transaction is modifiable, cacheable, posted or non posted, bufferable or non-bufferable [---0] Modifiable [---1] Non-Modifiable [--0-] Bufferable [--1-] Non-Bufferable [-0--] Write through write/read allocate [-1--] Write through no read allocate [0---] Non posted [1---] Posted
TxHdrRegion	[] carries information about which region the transaction belongs to
TxAtomicAttr	[000] Not Atomic [001] Exclusive [010] Atomic Store [011] Atomic Load [100] Atomic Swap [101] Atomic Compare [110] Locked [111] Reserved

### 4.2.13 OrderingModel

The OrderingModel[4] field defines the ordering models (different protocols) supported in the Fabric.

[-00]: No ordering required. Ignore Channel ID and Ordering ID

[-001]: Order only on ordering ID. Ignore Channel ID. Hence if two transactions have the same ordering ID they will be ordered regardless of the channel ID. No ordering between transactions with different IDs whatsoever.

[-011]: Order on both the channel ID and Ordering ID. That is order transactions that match on channel ID and ordering ID. No ordering on across channels of any kind. Always order within the channel on address overlap.

[-101]: Order transactions that match on Ordering ID and order transactions that match on address. Ignore Channel ID.

[-111]: Order transactions that match on Ordering ID and Channel ID. Also order transactions that match on Address overlap across Channel IDs.

[-110]: NA

[-010]: NA

## 4.2.14 OrderingId

This field informs Symphony network elements the ordering relationship between transactions with the same ordering ID. Note: Ordering ID may be different from Transaction ID.

## 4.2.15 ChannelID

This field defines the ordering relationship between transactions that have different OrderingIDs.

## 4.2.16 UserCrtl

This field captures the user defined signals.

## 4.2.17 Non-data-payload

This is meta data carried by the message.

## 4.2.18 Data

Write Request data or Read Response data.

## 4.2.19 Bytes Enables

This field indicates which bytes in the Data field are Valid bytes.

## 4.2.20 RESP

Response signaling (for response messages only).

## 4.3 ATP Data Link Layer Header

This section defines the header used by the data link layer to forward packets from one element in the fabric to another.

The data link header contains the following fields:

**Table 4-9 Data Link Header**

Fields	Description	Bit Range	Parameterizable
pkType	Packet Type	2	Yes
pkRoute	Self Routing Bits	Depends on network and system configuration	Yes
pkTID	Target ID	Depends on network and system configuration	Yes
pkQoS	Quality of Service	0-4	Yes
pkSec	Security level	0-2	Yes
pkVNID	Virtual Network ID	Depends on network and system configuration and requirements	Yes
pkID	Packet ID	Depends on system configuration	Yes
pkMult	Multicast Info	Depends on the number of multicast trees supported in the network	Yes
pkError	Error indication	0-4	Yes

**Table 4-9 Data Link Header**

Fields	Description	Bit Range	Parameterizable
pkSecFail	Security violation	2	Yes
pkTier	Informs the system about packets belonging to different forwarding types	Depends on the system configuration	Yes
pkPri	[] Informs the network the relative priority of the packet	Depends on the system configuration	Yes
pkSID	Source ID of the packet	Depends on the system configuration	Yes
pkTxnResp	Indicates whether the response is Ok or errored	2	
pkDPresp	Indicates the status of the Data payload	2	
pkDummy	Indicates the end of the packet	1	

### 4.3.0.1 Field Descriptions

### 4.3.0.2 pkType

Table 4-10, defines the different types of packets supported by Symphony. A normal packet can be read/

**Table 4-10 Packet Type.**

Packet Type	Definition
00	Normal Packet
01	Credit Packet
10	Control Packet
11	Reserved

write or Atomic transaction.

Control packet is packet is separate from the normal packet and can be used to convey information between two Network Endpoints and the network elements. For example, an unlock packet is a control packet that will unlock the path on which it traverses.

Credit packet carries credit information between the Src and Dest. This packet is not visible at the CTL layer.

- CTL layer mapping: 1) All types of Request/Response transaction info is mapped into the Regular Packet field [00]
- 2) Control transaction is mapped to the Control Packet [10]
  - 3) Credit packet info is mapped to credit packet

### **4.3.0.3 pkRoute**

This field is the self-routing header for the packet. If the route lookup is in the switches, then this field is not required. The route field would be of the same length for the Fabric, i.e., no matter whether the packet has to travel a large number of hops or a small number, the route field will have the same number of bits. Route bit will be protected.

### **4.3.0.4 pkDestID**

This field has the Destination ID for the packet. The destination field is necessary for the Target to confirm whether it is the intended destination for the packet.

### **4.3.0.5 pkSrcID**

This field identifies the Initiator ATU associated with the Transaction. This field is parameterizable and will be configured at the time of setting up the Interconnect.

### **4.3.0.6 pkTier**

Tier field is used by the message to determine the forwarding policies of the message. Each tier may map to a different virtual channel. Packets belonging to one tier should not block packets belonging to another tier.

### **4.3.0.7 pkPri**

This field defines the absolute priority of the packet with respect to other packets. The pkPri can have variable encoding when implemented, but the lower valued number will have higher priority than the higher valued number. Priority can also be encoded using the one-hot concept, but again, the lower value will have higher priority than the higher value.

### **4.3.0.8 pkQoS**

This field defines the QoS level of the packet. The fields are defined in terms of QoS requirements of the different traffic types within the SoC.

The fields can be overloaded to indicate strict priority as well.

**Table 4-11 Packet QoS.**

pkQoS[3:0]	Definition
0x0	Latency Sensitive – High Priority
0x1	Isochronous (real-time traffic) – Mid Priority
0x2	Bandwidth Guaranteed Traffic – Mid Priority
0x3	Bursty with urgency – Mid Priority
0x4	Low priority with starvation protection
Others	Reserved

### **4.3.0.9 pkSec**

This field defines the Security level of the packet.

**Table 4-12 pkSec Definition.**

pkSec	Definition
0x00	Normal Transaction
0x01	Secure Transaction (Carries the NS bit)
Others	User Defined Secure Level

### **4.3.0.10 pkVNID**

This field defines the Virtual Network ID associated with the packet. The number of Virtual Networks supported on a link can be restricted.

### **4.3.0.11 pkID**

This field carries the packet ID. This is an optional field, and will be used if there is a use case (like per packet ack/nack handshake).

### **4.3.0.12 pkMulti**

This field defines the multicast label used in the network to support multicast traffic.

### 4.3.0.13 pkError

**Table 4-13 PkError Definition.**

Error[5:0]	Error Type
[----0]	No Error detected
[----1]	Error detected
[---11]	Transport error decode
[--1-1]	Path disconnection error
[--1--1]	Slave Error
[--1---1]	Decode Error
[1----1]	Exclusive Transaction Error (ExOkay)

Error Detected: All other errors that are not explicitly spelled out

Transport error: Error that is caused at the transport level such as a mis-routed packet. This error captures all errors reported by the transport network elements

Access violation: If the packet attempted to access address it should not have.

Path disconnection error: Packet sent along a path that was either powered/clocked down.

Some of error reporting and management will not be visible at the CTL layer.

### 4.3.0.14 SecFail

This field indicates the type of security violation.

**Table 4-14 SecFail Definition.**

SecFail	Security Violation Type
0x00	No violation
0x01	Violation

### 4.3.0.15pkTxnResp/pkDPrep:

TxnResp	
0x00	Okay
0x01	Error

TxnResp	
0x10-11	Reserved

### 4.3.1 Packet Side-band signals

*Table 4-15 Data Link Header*

Signal	Bits	Description	Optional
Valid	1	Indicates a valid PHIT if both valid and ready are asserted (per VC)	
Ready	1	Ready signal for valid-ready handshake (per VC)	
First	1	Indicates first PHIT	
Last	1	Indicates last PHIT	
ECC or parity		ECC (bits dependent on data-width) or parity (a bit per parametrized data-width) calculated per PHIT	Yes
Panic	1	When asserted indicates raised priority of the port/VC (per VC)	

## 4.4 Credit Packet

The message for credit packet shown in Table 4-16 below is optimized for the case where there is a separate control Fabric over which credit packets sent. If credit packets are sent over same Fabric as normal transactions then the credit packet will use the same data link header as normal packets but the credit information would be contained in the non-data-payload section of the packet.

*Table 4-16 Credit Packet*

Fields	Description	Bit Range	Parameterizable
Route	Route bits	Depends on system configuration	Yes

**Table 4-16 Credit Packet**

Fields	Description	Bit Range	Parameterizable
SrcID	Source Identifier	Depends on system configuration	Yes
DestID	Destination Identifier (Target Identifier)	Depends on system configuration	Yes
QoS	Quality of service	0-4	Yes
VN	Virtual Network the message belongs to	Depends on system configuration	Yes
CrMsgType	Type of Message – identifies a credit req/ grant, credit resync message	4	Yes
CrMsgID	Credit message identifier	1-8	Yes
CrCredit	Number of credits requested or granted	10	Yes

#### 4.4.1 CrMsgType[3:0]

This field defines the type of credit message. The types are given below:

**Table 4-17 CrMsgType.**

CrMsgType[3:0]	Description
0000	Stop transmitting
0001	Credit request
0010	Credit grant
0011	Resynch credits
others	Reserved
1111	Resume transmission

## 4.5 Control Packet

The packet format and fields define a common control packet that can be used for carrying control information. The control packet can be used to carry error information, interrupts, other control information.

If the control packet is sent over the Fabric which is also used by normal transactions, then data link header would be the same as that for normal transactions and the control information would be part of the meta data (non-data-payload). If on the other hand it uses a dedicated Fabric, the optimized header fields are listed in Table 4-18 below.

**Table 4-18 Control packet Data Link Header**

Fields	Description	Bit Range	Parameterizable
Route	Route bits	Depends on system configuration	Yes
SrcID	Source Identifier	Depends on system configuration	Yes
DestID	Destination Identifier (Target Identifier)	Depends on system configuration	Yes
PacketID	Packet ID with which the control packet is associated with.	Depends on system configuration	Yes
VN	Virtual Network the message belongs to	Depends on system configuration	Yes
MsgType	Type of Message – Error, Control, Status	4	
MsgID	Control message identifier	1-8	Yes
MsgAddr	Message Address	8-48	Yes (A config space can be defined to store messages, and this is address for that)
Message	Message Data	0,8,16	Yes

## 4.5.1 MsgType

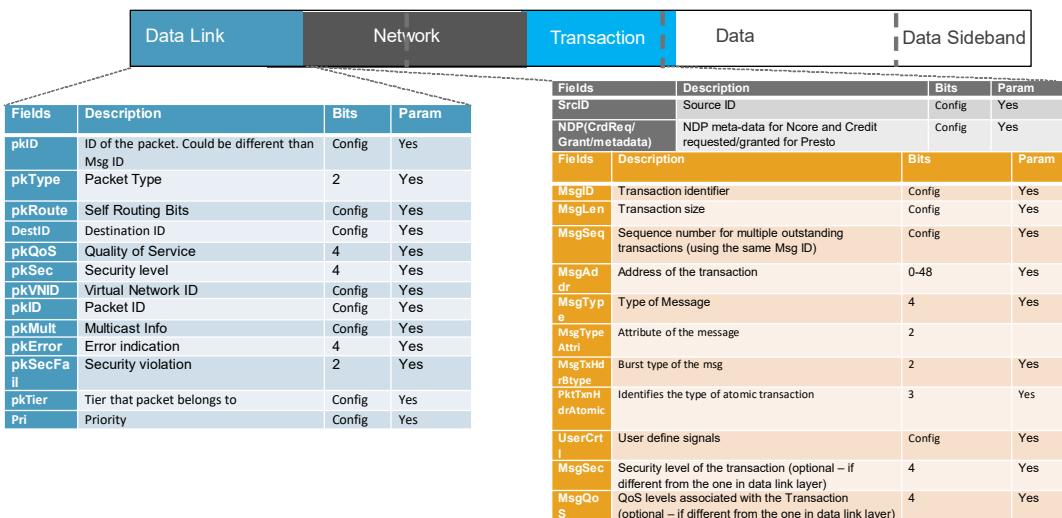
**Table 4-19 MsgType.**

MsgType[3:0]	Description
0000	Interrupt
0001	MSI – need more thought
0010	MPI – need more thought
0011	ACK
0100	NACK
Others	Reserved

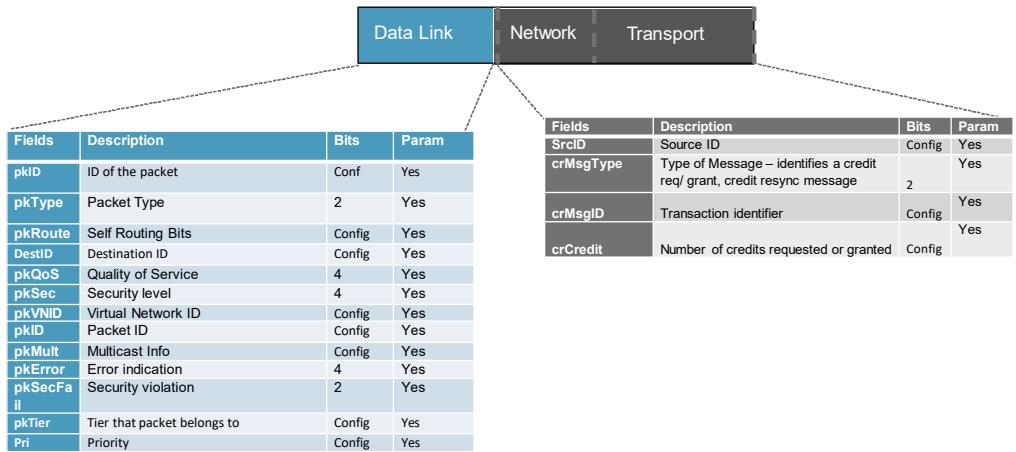
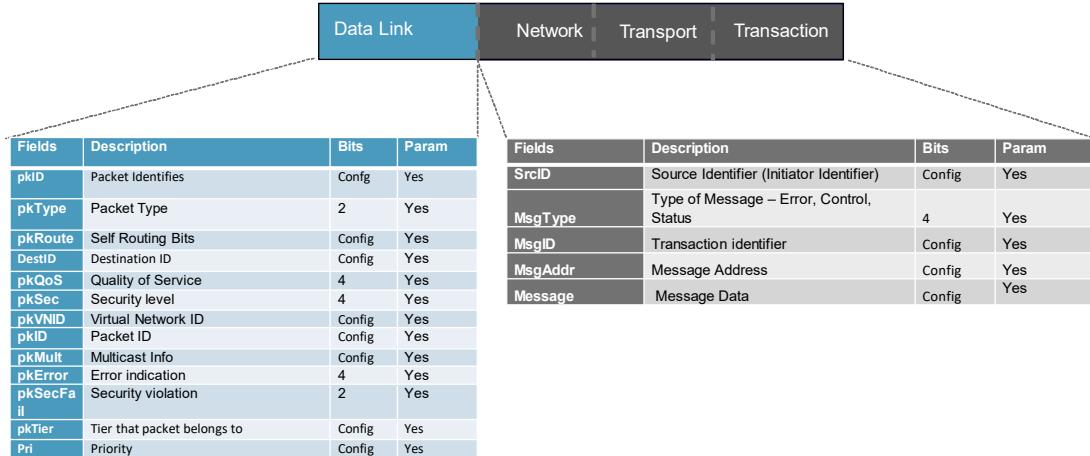
## 4.6 ATP Packet:

ATP Packet carries information required to make decisions at the data link layer, the network layer and the transaction layer along with the data payload. The figures below describe the packet format.

**Figure 4-2 Figure 1 Normal Packet**



**Figure 4-3 Figure 2 Credit Packet**

**Figure 4-4 Figure 3 Control Packet**

The data link layer packet header remains same for all packet formats (described in detail below). The Network and Transaction layer fields are fixed per packet type. So, given a packet type in a fabric, the *packet header size is fixed (determined by the parameters defining the packet format)*. If the credit and control packets traverse over a separate dedicated network, there are opportunities to reduce the packet header size.

Packet header will be uniform within a Fabric. Different Fabrics connected via adapters can have different sized packet headers, thus giving the system designer the opportunity to optimize the link widths.

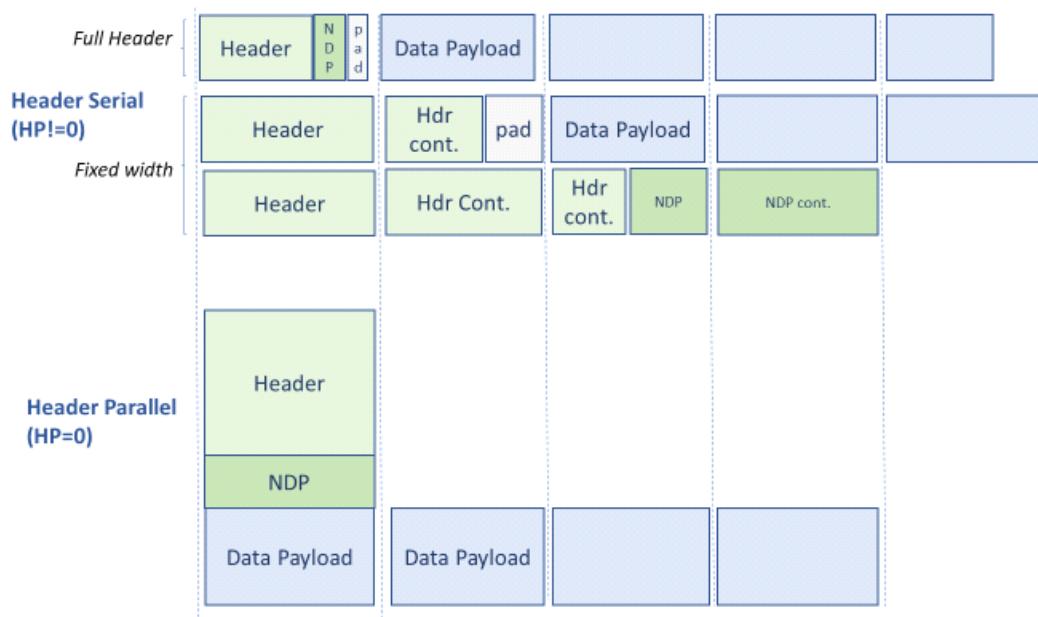
Different Fabrics within the Interconnect may have different packet sizes. The number of bits assigned to packet header fields may change and some fields may not be required all together.

## 4.7 Packet PHITs

A packet is made of PHITs (FLITs), where each PHIT is a single beat on the packet network. Header and data are never located in the same PHIT unless it is header penalty 0 mode.

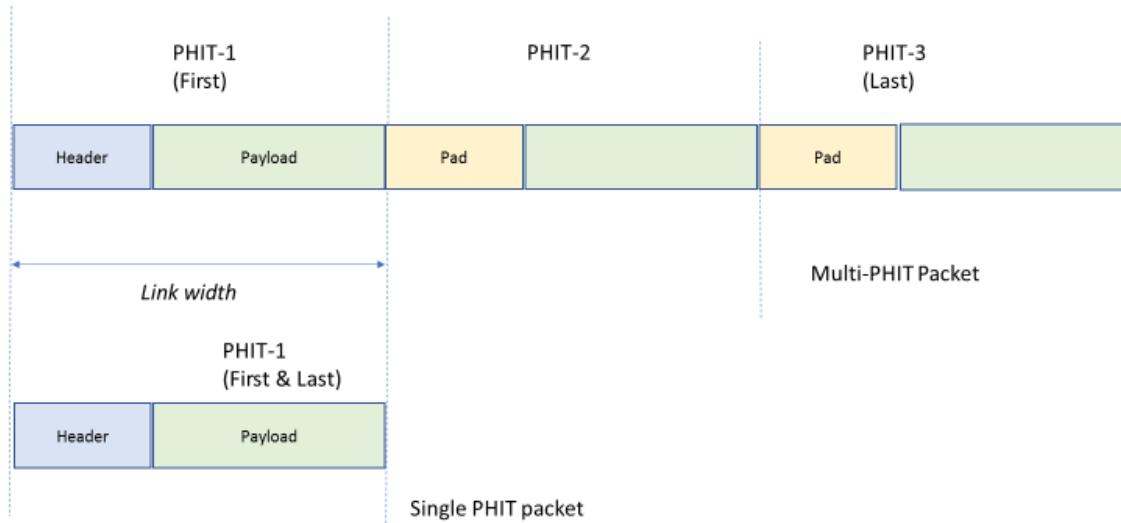
- ▶ The header may occupy one PHIT (first) or span over multiple PHITs.
- ▶ The minimum packet size is one PHIT, which carries both the header and the payload.
- ▶ For multi PHIT packets –
  - ♦ For header serial mode: payload is not packed in the same PHIT as header. The data PHITs contain the payload data and the data sideband, and follows the header PHIT(s).
  - ♦ With zero latency header: Header and payload is packed into the same PHIT. The PHITs after the first PHIT, has padding instead of the header.

**Figure 4-5 A Figure**



**Figure 4-6 Packet Header and Payload PHIT mapping**

## Packet Header Alignment – Zero Latency Header



Max Data size = Configurable.

Min Data size = 1B

Maximum PHIT size = Configurable

Minimum PHIT size = 1B

Note that Symphony ATP protocol allows for the packet to carry multiple different payloads based on the definition in the packet type and packet attributes, for example, Symphony can support the following configuration:

- ▶ Header Parallel: H-P0, PadP0-P0, PadP0-P0, PadP1-P1, PadP1-P1
- ▶ Header Serial: H-PadH, PadP0-P0, PadP0-P0, PadP1-P1 ...
- ▶ Fixed Serial: H1, H2-PadH2, PadP0-P0, PadP0-P0, PadP1-P1, PadP1-P1

## 4.8 Packet Error Handling

A packet error can be detected by -

- ▶ Initiator ATU – while generating the packet for a request, or in the received packet from the fabric.
- ▶ NOC Fabric – while passing a packet through the network packet error can be detected in the Firewalls. Pipelined switches will also detect errors. Adapters.
- ▶ Target ATU – while receiving a packet from the fabric, or generating packet for a response (or credit message).

- ▶ If no checking is done in the Fabric, then route bits should be configured such that errors do not result in packets circulating in the network indefinitely.

Any packet that is detected to have an error (different error types and checks) is marked as an errored packet. There is an error field in the packet header to capture the error type.

The errored packet is not dropped or responded back by the module that detects the error (it will generate an interrupt for the detected error) but is handled like a regular packet except for marking error in the packet header. This avoids the need for any extra handling for maintaining ordering for the errored packet (flows like a regular packet). For details see “error management spec”.

## 4.9 ACK/NACK

For symphony packet protocol, ACK/NACK acknowledgment can be enabled (optional) for every transaction sent out as a packet into the network, using control packets (refer the control packet section). A control packet (ACK/NACK) is generated for every packet received by a target (or a network node) except for the ACK/NACK control packets themselves, to be sent back to the source. The control packets could be sent on separate control network or piggy backed on the response packets.

ACK – packet was received by the target successfully without errors.

NACK – packet was received by the target with errors (like ECC or security errors), or could not reach the target (like target power domain is off).

There is no replay capability in the source to re-send a packet that receives a NACK (which can be supported with additional area cost of replay buffer).

Without ACK/NACK support:

For non-posted transactions- the error bits set in a response implies a NACK, otherwise an ACK (there are no packet drops in the network).

For posted transactions – the slave does not respond. When ack/nack is enabled, the Target will respond with an ack or nack.

## 4.10 Base Packet Definition

Symphony has a base packet definition concept, whereby each Fabric will have a defined packet format. That is, the packet format, for that Fabric will define the fields which maybe a subset of the fields defined in this chapter, their location in the packet header and their hardware interpretation. For example, the multicast field in the packet header could be interpreted as a multicast label for routing reasons or an indication that the packet is a multicast packet and the route field represents the multicast label.



# 5

# Symphony Switch Architecture

## 5.1 Introduction

Symphony will support the following type of switches:

1. Flow Through Switches
2. VC-Aware Flow Through Switches
3. Buffered VC Switches
4. Deeply pipelined Switches

The objective is to offer multiple types of switching elements with different area, power, performance and features in order to enable Arteris to construct Domain Adaptive Fabrics which are power, performance and area optimized to a given application.

This chapter specifies the architectures of the Symphony switching elements. The specification of the architecture of a switching element comprises of:

- ▶ the different sub-components of the switch
- ▶ relationship between the different subcomponents
- ▶ signaling and flow control capabilities
- ▶ scheduling and arbitration schemes are described in the section on Scheduling.

This chapter does not delve into the micro-architecture details of the switches. However, the chapter does include implementation notes that allude to restrictions or lack thereof that are placed on first release of Symphony.

This chapter assumes that all traffic flowing through fabric structures is in the form of units of information called *packets*. A packet is comprised of an amount of information, called a *message*, being sent by the source device to the destination device as well as additional information that helps steer the packet to its destination. A complete definition of these packets is part of the definition of the Transport Protocol, which identifies the fields within these packets that are essential to enabling the various capabilities of the switches and to the overall operation of the fabric. The Transport Protocol is the subject of section on Arteris Transport Protocol.

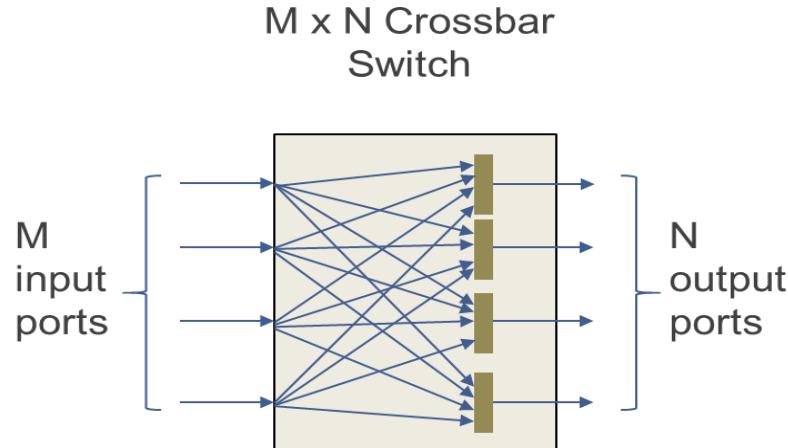
## 5.2 Symphony Basic Flow-Through (FT) Switch

- ▶ Simple M x N Crossbar switch
- ▶ No internal buffering: Combinational data paths from input to output
- ▶ An Arbiter per output port
  - ◆ Support for RR, PRR, WRR, FCFS scheduling algorithms

- Ability to “hold” the selection made based on Request until the packet is completely transmitted

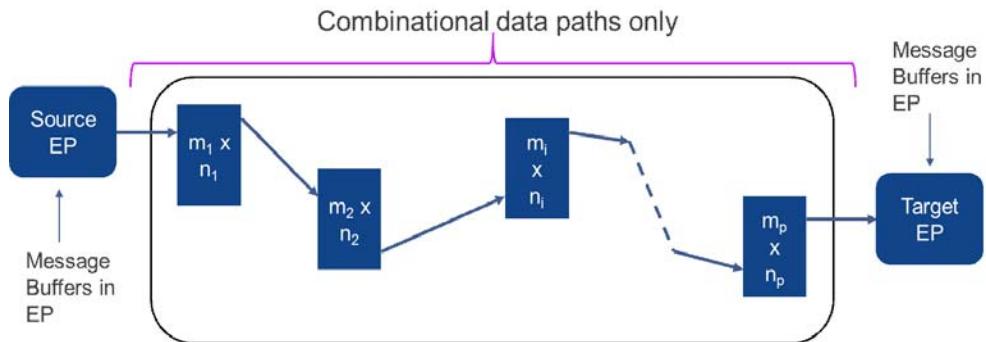
- Selectable port widths

**Figure 5-1 MXN Crossbar Switch**



### Symphony Fabric using FT Switch Elements

**Figure 5-2 Fabric with Flow-Through Switches.**



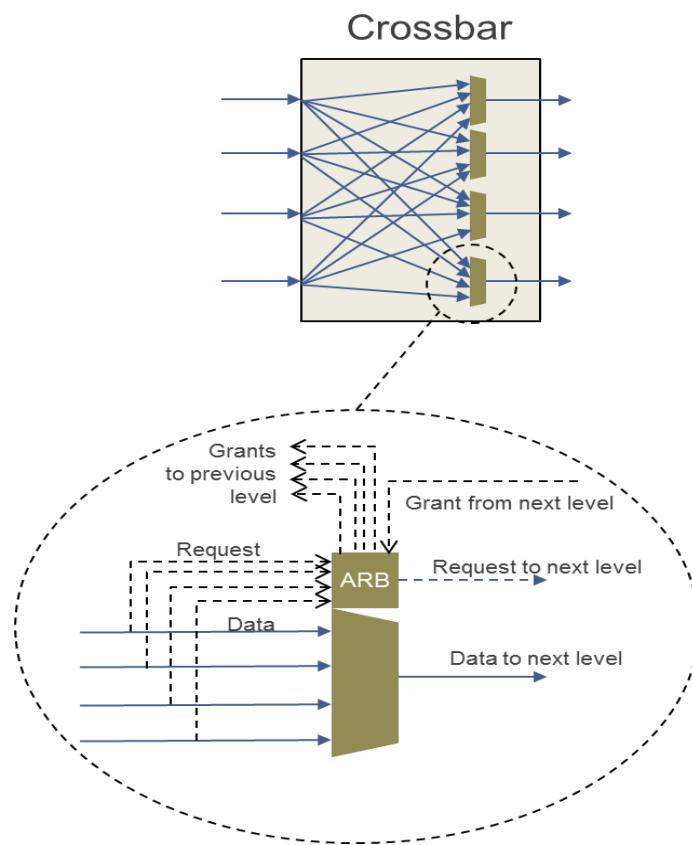
- Messages from Source End Point to Target End Point pass through a switch fabric comprised of a sequence of  $p$  crossbar switches  $S_1$  through  $S_p$
- Each switch  $S_i$  has  $m_i$  input ports and  $n_i$  output ports. Thus  $S_1 = m_1 \times n_1$ ;  $S_2 = m_2 \times n_2$ ; etc. Note that  $n_i > 0$  for all  $i$ .
- The path is static and can be pre-calculated based on the chosen inter-switch connection topology

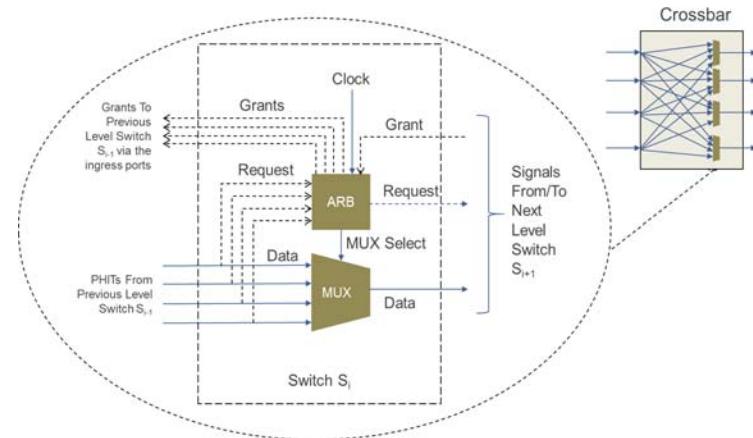
## 5.3 Symphony FT Switch Element Architecture

- Simple  $M \times N$  Crossbar switch
- No internal buffering: Combinational data paths from input to output

- ▶ An Arbiter per output port
  - ♦ Support for Rotate, RR, PRR, FCFS, Priority FCFS, Priority Rotate scheduling algorithms
  - ♦ Ability to “hold” the selection made based on Request until the packet is completely transmitted
- ▶ Selectable port widths

**Figure 5-3 Switch Diagram with Zoom-in of Output Port Arbiter.**



**Figure 5-4 Switch Crossbar with Zoom-in of an Output Port**

### 5.3.1FT Switch Element Internal Architecture

- ▶ The switch can look at the route field or a set of fields and decides how to route the packet either by using a look up table or by simply using the bits to route the packet to the egress port. The restriction is that all the fields that are used in determining the egress port have to be in the first beat of the packet.
- ▶ Each egress port ( $PE_{S_i}$ ) of the switch has its own arbiter (ARB)
- ▶ Each arbiter implements one of the following arbitration policies: Rotate (ROT/Priority ROT), Round-robin (RR), Priority RR (PRR), First-come-first-serve (FCFS/Priority FCFS)
- ▶ The arbiter chooses one of ingress ports ( $C-PI_{S_i}$ ) currently requesting access to its egress port, PE, and
  - ◆ Asserts a Grant to  $C-PI_{S_i}$ . The Grant propagates thence to  $S_{i-1}$  connected at the chosen ingress port,  $C-PI_{S_i}$ , of  $S_i$ .
  - ◆ Asserts a Request to  $S_{i+1}$  connected to this egress port, PE
  - ◆ Sets the MUX Select to allow the signals of the chosen ingress port,  $C-PI_{S_i}$ , to flow to the egress port, PE
  - ◆ The PHIT contents along with “Route” (which indicates the  $PE_{S_{i+1}}$  of the next switch  $S_{i+1}$ ) is presented to the connected  $PI_{S_{i+1}}$
- ▶ The selection at this PE is “held” until the whole packet passes through it. After that, the arbiter logic arbitrates again among the next set of requesting ingress ports.

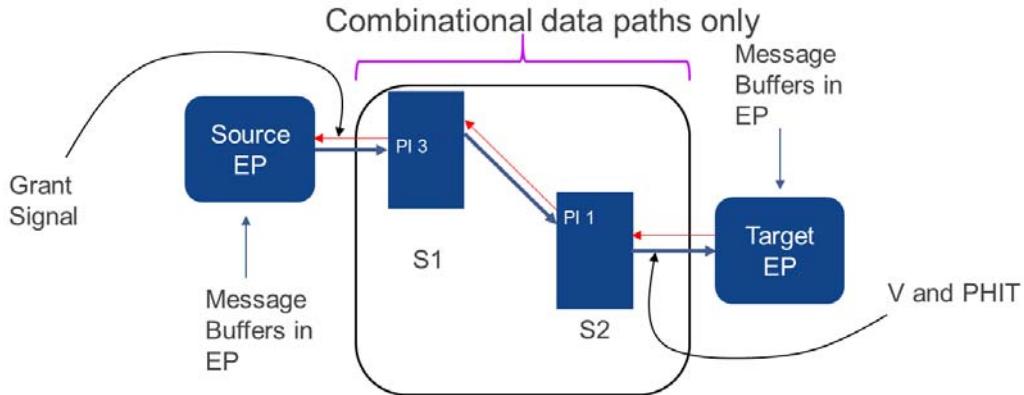
### 5.3.2Target Behavior for FT Switch

- ▶ Grant signal is asserted at a target device indicating the availability of buffering for a single FLIT worth of message (i.e. 1 MAX\_PHITs number of PHITs). It is negated otherwise.
- ▶ Once asserted, the Grant signal is kept asserted until the target receives the message PHIT with asserted Last signal indicating the end of the current message. Thus, the target is expected to be able to receive entire messages even if they are multi-PHIT.

- ▶ The target may continue to assert the Grant signal after the end of the current message if it has buffering for another message

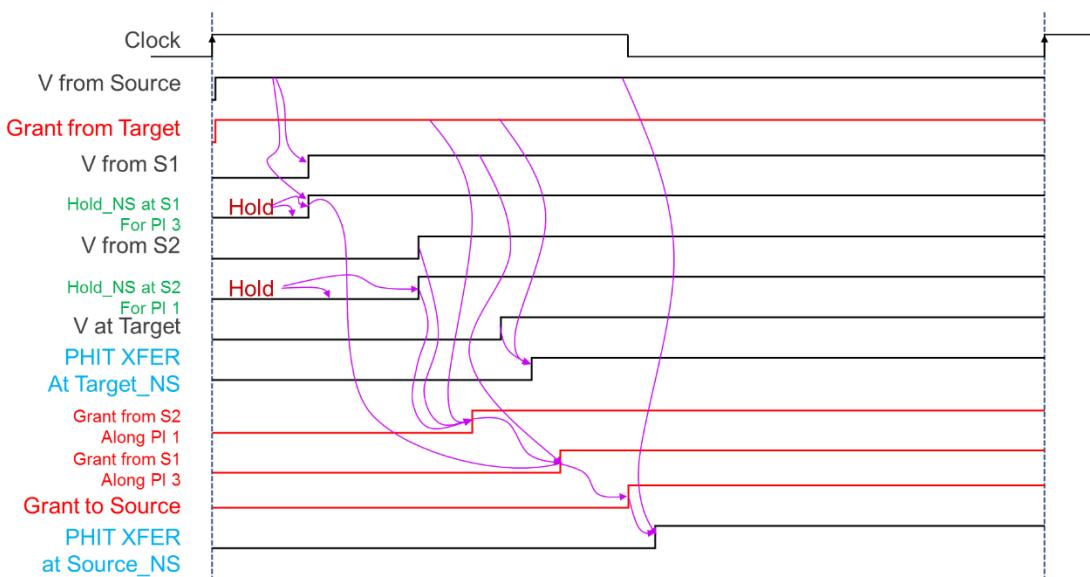
### 5.3.3 Timing diagram for PHIT propagation from Source to Target in FT Fabric

Figure 5-5 Example: PHIT propagation from Source to Target in a two deep FT Fabric



- ▶ Valid (V) along with the PHIT flow from Source to Target using Route information
- ▶ Grant flows from Target to Source along the path established by V and PHIT. Note that the grant is always asserted if the target is ready to accept a PHIT.
- ▶ Timing diagram is shown below
  - ◆ Diagram assumes the PHIT encounters no contention at S1 and S2

\*\_NS are “next state” signals that captured in a state register at the end of the cycle

**Figure 5-6 End-to-end Switch Arbitration Timing Diagram**

### 5.3.4 Null PHIT transfer in Flow-through fabric

- In a flow-through fabric a valid PHIT is considered transferred across two network elements that can buffer the PHIT in a single cycle
- The PHIT travels between multiple cascaded pairs of link partners
- During a cycle, a valid PHIT transfer from a Sender block to a Receiver block is detected and registered simultaneously, and independently, by these link partners. A valid PHIT transfer occurs during a clock cycle ONLY when:
  - The Sender asserts the Valid PHIT signal over the link and simultaneously receives an asserted Grant over the link
  - The Receiver asserts a Grant signal over the link and simultaneously receives an asserted Valid PHIT signal over the link
- When either the Valid from the Sender or the Grant from the Receiver is missing during a clock cycle, a valid PHIT is not considered transferred over the link. Such a non-transfer with negated Valid but an asserted Grant is referred to as a “null” or a “bubble” PHIT.

Note that such a bubble PHIT transmission can occur in the middle of a multi-PHIT message transfer.

### 5.3.5 Managing locked transfers

Although locked transactions have been deprecated for most protocols as they impact the performance of the Fabric drastically especially large fabrics, however, Symphony would support locked transfers for legacy reasons. AHB supports locked transfers. The way Symphony will support locked transfers is not de-assert last on the sideband signals until the transfer is over. The ATP protocol has an independent last in the packet format that will indicate to the end processing unit such as the ATU packet delineation boundary.

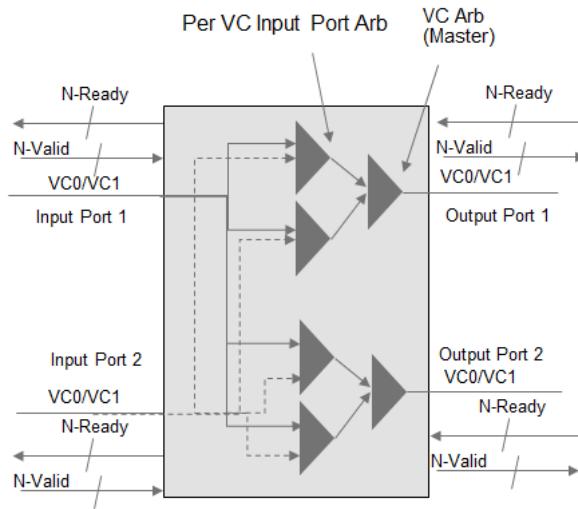
## 5.4 VC-Aware Flow Through Switch

Virtual Channels allow multiple messages on different VCs to share the common physical channel. PHITs from different VCs are allowed to multiplex onto the same channel. However, PHITs belonging to different packets but from the same VC are not allowed to interleave. That is, once a multi-PHIT packet transmission has started, no other packet from the same VC can be interleaved with the packet that has just started transmission.

VC Flow Through switch architecture adds virtual channels to the Flow Through switches described previously in this chapter. The following are added to the base Flow Through architecture:

1. Two level hierarchical arbiter is added to each egress port. The first level arbiter is a per VC arbiter which arbitrates between the different inputs that have a valid packet to send on that VC. The second level arbiter is called the “Master” arbiter, and arbitrates between the valid VCs on the egress port.
2. The state of each VC is reflected by a Valid and Ready signal. On each ingress and egress port a vector of Valid and Ready signals is added to reflect the state of that VC. In the flow through setup, there can be only one Valid ever be asserted per port.
3. When a valid is asserted on an input port for a particular VC, the route field in the packet header is used to route the control signal to the appropriate egress port first level arbiter (per VC arbiter). The first level arbiter locks if the input port wins arbitration. The arbiter only locks on the first PHIT and releases the lock on the last PHIT of the packet, so as not allow PHITs from other input ports to interleave on the same VC. The second level arbiter (Master), then chooses between the valid VCs. It should be noted that the first level arbiter will be referred to as the *VC allocation arbiter* and arbitrates once a packet, whereas the Master arbiter arbitrates on a per PHIT basis.
4. It should be noted, that in the VC-Aware Flow Through switches, the Master arbiter chooses the VC to transmit the PHIT from, without taking into account the Ready signal from the downstream switch.

In Figure 5-7 on page 68, the VC-Aware 2x2 switch with 2 VCs per port architecture is detailed. There are 2 valid and 2 ready signals per port in this case. Since there are only two VCs per port, the number of input port arbiters per egress port is two and there is only one VC arbiter per egress port as shown.



**Figure 5-7 VC-Aware Flow Through Switch**

---

**Note**  
Restriction: VC-Aware Flow Through switch can only be used when the Master Arbiter is a Priority Arbiter.  
See section on VC-Shutout.

---

### 5.4.1 Ready Aware VC-Aware Flow Through Switch

Ready Aware VC-Aware flow through switch is a variant of the VC-Aware Flow Through switch. In the Ready Aware VC Flow Through Switch, the arbiter takes the status of the VC at the downstream network element into account before choosing the VC. That is, the VC becomes eligible for arbitration by the Master arbiter, if the following conditions hold:

1. The VC has a valid PHIT to send.
2. The Ready for the corresponding VC is asserted by the downstream network element on the port.

Because of timing restrictions, this switch can only be used as the last switch along the path before the T-ATU, or a VC pipelined buffer, or any other VC enabled network element that can buffer the PHIT/ Packet.

## 5.5 Soft Locking Arbiter (Flit Biasing)

There are two key benefits for implementing virtual channels. These are:

1. Deadlock avoidance.
2. Reducing Head Of Line Blocking (HoL).

Both of the above benefits are important to allow topologies such as meshes and rings and also in improving the performance of the network.

VC scheduling as described in Section on scheduling, the VC arbiter re-arbitrates every cycle to allow for different VCs to make forward progress based on the arbitration scheme and the bandwidth allocated to

that VC. While this scheme is fair in distributing bandwidth, it introduces bubbles in packets. Note that, with VC wormhole switching bubbles in packets are inevitable.

We introduce Flit biasing, the objective to be able keep the intra-PHIT jitter between PHITs belonging to a packet to the minimum.

---

— Note —

Packet Interleaving Definition: Once a packet starts transmission on the link, no other packet can be multiplexed on to the link, until the current packet is done transmitting, even if the other packet is on a different VC.

---



---

— Note —

Flit Interleaving Definition: Once a Flit starts transmission on the link no other Flit can share the link until the Flit is done, even if the other Flits belong to different VC. Once the Flit is done transmitting, others can transmit.

---

## 5.5.1 Soft Locking Mechanics for Flow Through VC aware switches

Soft locking has two components. One deals with how and when the VC arbiter re-arbitrates and the other is how the arbiter updates its state when it re-arbitrates.

The VC arbiter is modified to re-arbitrate only if the grant was not received in the previous cycle. If the grant for VC was received in the previous cycle the arbiter keeps on servicing the current VC. The arbiter will only re-arbitrate when there was no grant received in the previous cycle or the last PHIT for the current packet was serviced, or if there was bubble in the packet. The conditions for soft locking are stated below:

- ▶ Condition 1:
  - ◆ Cont. servicing  $VC_i$  in cycle $t$  iff  $\neg\{Grant \text{ was received for that } VC \text{ in cycle}_{t-1} \text{ \&\& } !FirstPit \text{ \&\& Valid PHIT}\}$ .
- ▶ Condition 2:
  - ◆ Arbitrate if the above is not true.

The above condition states that the arbiter will keep servicing a VC, if a grant was received in the previous cycle and the VC has a valid PHIT. If either the VC does not have a valid PHIT (bubble) or it is the first PHIT of the packet or there was no grant issued in the previous cycle, only then the arbiter will re-arbitrate.

Soft locking, does not prevent packets on other VCs from making forward progress. Packets from other VCs can make progress whenever the arbiter re-arbitrates.

It is desirable that the arbiter finish servicing the packet in the order they got interrupted, that is, finish servicing partially serviced packets first before servicing new packets. However, implementing a priority order may require more state to be stored in the arbiter and will increase area and power.

---

— Note —

We are also investigating another approach where soft lock would be established for the VC that receives a grant. The arbiter will service another VC only if there is a bubble in the packet that is being currently serviced.

---

## 5.5.2 Simulation Results

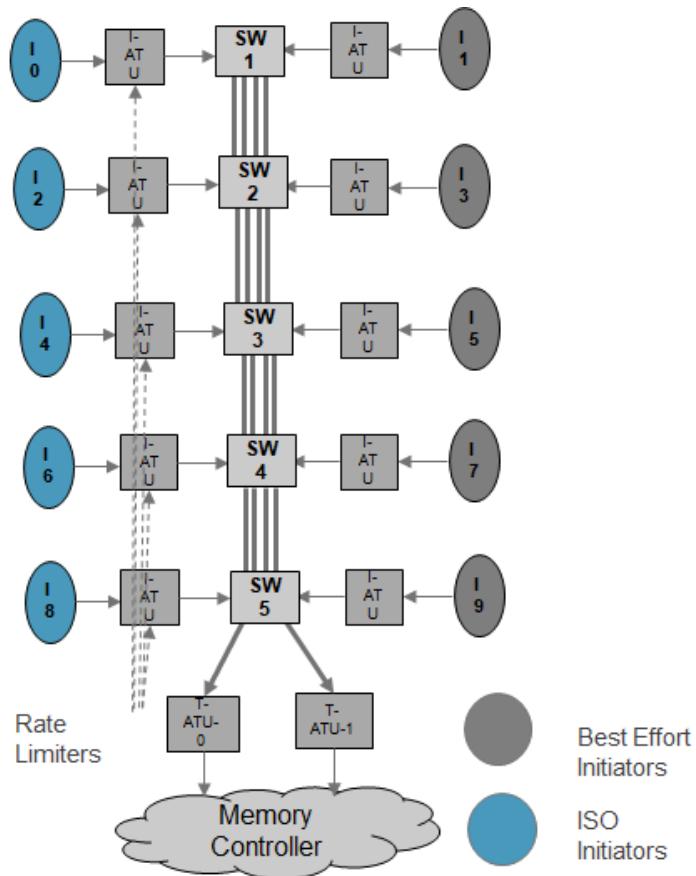
The network shown in Figure 5-8 on page 70 was simulated to investigate the above described scheme. The network consists of 10 Initiators and two Targets. The initiators are either Isochronous and Best

Effort. Traffic generated from Isochronous initiators is given priority over traffic generated by best effort initiators.

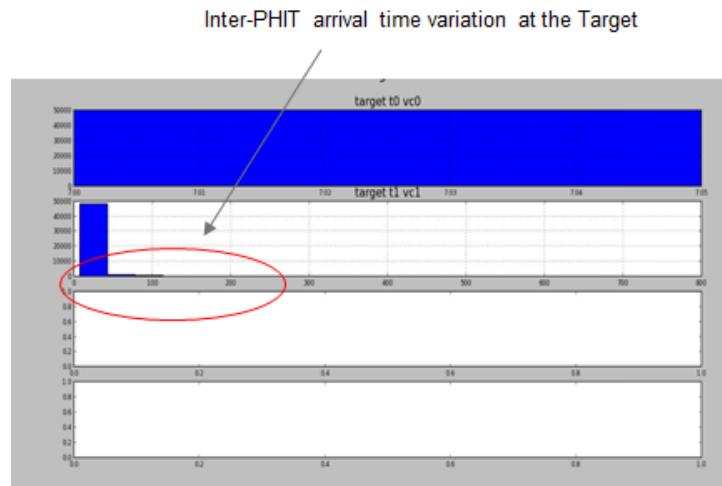
Two experiments were performed. One without soft locking and one with soft locking enabled. The simulations measured the difference between the time the first PHIT of the packet was received at the target and when the last PHIT was received. In other words, we plot the Intra PHIT jitter.

The performance results are presented in Figure 5-9 on page 71 and Figure 5-10 on page 71 below.

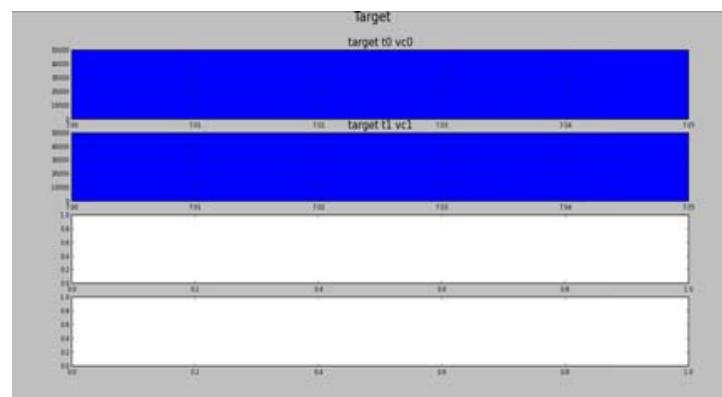
**Figure 5-8 Network simulated to investigate Soft Locking.**



**Figure 5-9 Inter PHIT arrival time variation at the Target without any Soft Locking.**



**Figure 5-10 Inter PHIT arrival time variation at the Target with Soft Locking.**

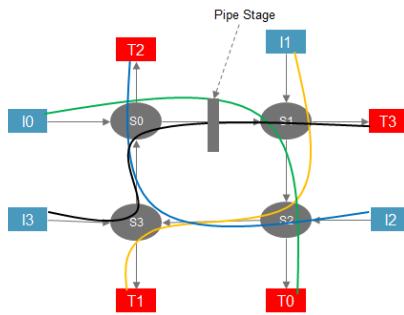


Each packet consists of 8 PHITs. Notice that with Soft Locking, the arrival time difference at the Target between the first PHIT and the last PHIT of the packet is 7 cycles. Which means that the packet is kept intact and not broken up. For high priority packets, it is expected that the PHITs belonging to a packet will be kept together, however, the above results show that with FLIT biasing this is also true for lower priority traffic as well.

Contrast this result with the one without Soft Locking. The arrival time difference between the first PHIT and the last PHIT of the packet at the Target varies over several cycles indicating that the network introduced bubbles in the lower priority packets. The network without Soft Locking will naturally keep the PHITs belonging to the higher priority packets together.

### 5.5.3 Deadlock Avoidance

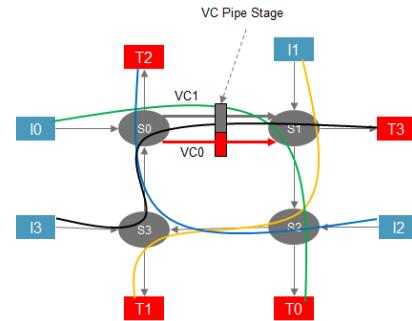
Deadlock avoidance is one of the key features of virtual channels. Consider the network shown in the Figure 5-11 on page 72 below (More details are presented in the Chapter on Routing). The network below is prone to deadlocks even when all the flows have no cyclic paths. The reason is because although the flows are non-cyclic, they need resources that may be held by other flows and in turn create a resource dependency cycle.



**Figure 5-11 Simple network that with acyclic flows can cause deadlocks.**

Virtual Channels help remove the deadlock by providing an alternate path for the blocked flows to make forward progress. The deadlock in the network shown in Figure 5-11 on page 72 is removed by adding a virtual channel on the link between S0 and S1 as shown in Figure 5-12 on page 72 below.

**Figure 5-12 Deadlock removed by adding a Virtual Channel between S0 and S1.**

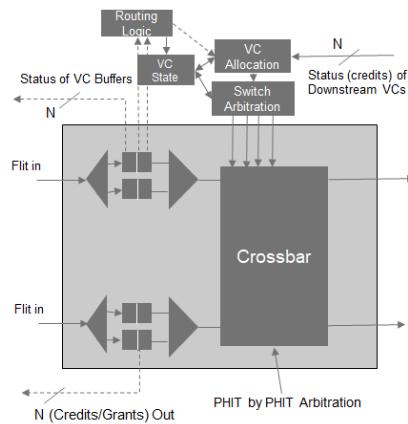


Because the above described Soft Locking scheme does not prevent other VCs from making forward progress, it therefore does not deadlock. However, if we had a scheme that prevented for some reason other VCs from making forward progress, then adding VCs would not resolve deadlocks.

## 5.6 Fully Buffered VC Switch

The canonical virtual channel switch (router) architecture consists of input buffers where Flits from the previous stage are buffered, routing logic that determines the forwarding port on the switch and the requested Virtual Channel, a virtual channel arbiter, switch arbiter and a crossbar switch to finally transmit the Flit from the input port to the output port. This canonical architecture is shown in the Figure 5-13 on page 72 below.

**Figure 5-13 Canonical VC Switch**



The canonical switch function is split into four distinct phases. These phases are: 1) Route determination; 2) Virtual channel allocation; 3) Switch allocation; and 4) switch traversal. Phases 2 and 3, involve control path whereas phase 4 involves the data path.

In the virtual channel buffered switch each virtual channel has a dedicated input queue. As the Flits arrive from the upstream switch, they are buffered in the appropriate Virtual channel input queue. The input queues service Flits in a FIFO order.

As the first Flit of the packet arrives at the switch, the route field in the packet header is decoded by the route logic. Once the decode is done and the egress port and the associated virtual channel determined, the state of the Virtual Channel is updated accordingly. This phase is route determination phase.

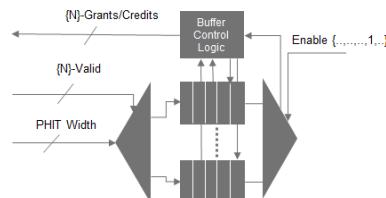
In the VC allocation phase, inputs present their desired output requests to the VC channel allocator. The VC channel allocator arbitrates between different valid requests to determine the best input-output VC match.

Once VC match has been made, the Switch then arbitrates to connect the appropriate input port to the output port. This phase is called the switch arbitration phase. And finally, in the switch traversal phase, the PHIT traverses the switch and is transmitted on to the next switch downstream.

## 5.6.1 Input Virtual Channel Buffer Block

The Input Virtual Channel Buffer block is shown in Figure 5-14 on page 73 below.

**Figure 5-14 Input Virtual Channel Buffer Block**



- ▶ The  $\{N\}$  bit valid vector on the ingress indicates that there is a valid PHIT on the link and to which VC it belongs to. Only one Valid bit is set.
- ▶ Each VC has a dedicated FIFO. (Note: implementation may decide to share the data buffer between the VCs -- see “Advance Buffer Management” below)
- ▶ The input demux routes the valid control signal to the appropriate VC FIFO.

- ▶ Message-In signal is sent to the appropriate VC FIFO.
- ▶ A ready signal is asserted if a free buffer is available. The assertion of the ready signal and the valid signal confirms PHIT transfer.
- ▶ The Buffer Control Logic (BCL) block updates the available buffer count for the VC. The BCL updates the status of VC buffer availability whenever there is a change in buffer occupancy, either when a PHIT is queued or dequeued.
- ▶ BCL block sends an update to upstream switch when there is a change in the buffer occupancy of the VC buffers.
- ▶ If link level credits are used, then N field vector update is sent to the credit count at the upstream switch. The credit update only needs to be sent when a PHIT is dequeued from the buffer if the credit counter at the upstream node is set to max credits when initialized.
- ▶ If a Ready/Valid interface is used then an N bit vector is sent back indicating the acceptance of the PHIT.
- ▶ Dequeue from the VC buffers happens when the Enable signal for that VC is asserted.

### 5.6.1.1 Advance Buffer Management

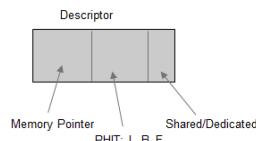
VCs on the same link have what is called negative correlation between them. That is if one VC on the link is heavily loaded, the other VCs on the same link would most likely be less loaded.

Dedicated VC buffering on the input port of a switch is easier to implement but results in underutilization of overall VC buffering available at the switch. The greater the number of VCs the larger would be the underutilization.

To improve buffer utilization and reduce overall cost or improve performance for the same level of buffering, the shared buffer approach is used to reduce the overall buffering cost for the same performance level. Shared buffer approach is shown in Figure 5-16 on page 75.

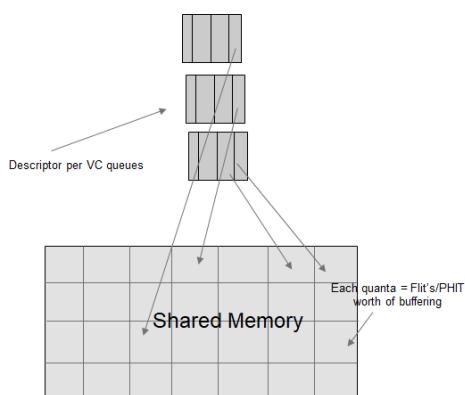
In the shared buffer approach buffers the FLIT/PHITs from the input port into the shared buffer. Buffer memory is segmented into dedicated and shared buffers. As the FLIT/PHIT comes into the input port, it is buffered and a descriptor is created. The switch checks the dedicated buffers first, for availability before checking the shared buffer availability.

The descriptor contains the memory address where the FLIT/PHIT is stored, the type of FLIT/PHIT (First, Body, Last), and whether the buffer is from the shared pool or a dedicated buffer as shown in Figure 5-15 on page 74 below.



**Figure 5-15 Descriptor that is queued in the VC queue**

Once the descriptor is dequeued, the corresponding PHIT/FLIT is read from the memory and sent across the switch.

**Figure 5-16 Shared memory with Descriptor VC queues.**

There is single shared counter that keeps track of available shared buffers and set of dedicated counters that keep track of the usage of dedicated buffer pool. The number of counters for dedicated buffers is equal to the number of VCs supported on the input port. The upstream switch is informed when a dedicated or shared buffer becomes available. The tracking of shared and dedicated counters is done at the upstream switch.

It should be noted that the memory at the very least is dual ported, that is should allow a read and write within the same cycle when NxM data path crossbar is used. If an NVxMW data path crossbar is implemented, then multiple reads can happen in a single cycle.

---

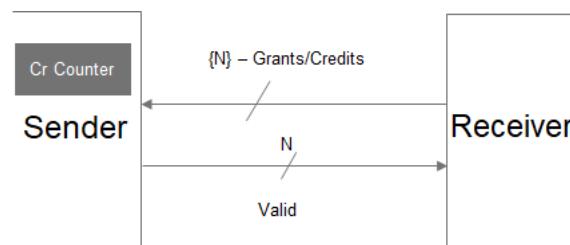
Implementation	Initial instantiation of the Buffered VC switch does not need advance buffer management
----------------	---

---

## 5.6.2 Flow Control Rules

Let  $CR_{VCi}$  be defined as the number of credits available for  $VC_i$  to transmit PHITs on the link and let " $m$ " be defined as the number of PHIT buffers allocated to  $VC_i$ .

The flow control rules for the buffered VC switch are as follows for VC switches with dedicated input buffers.

**Figure 5-17 Flow Control for Buffered VC switch**

- ▶ The Sender only sends a PHIT if the  $CR_{VCi} > 0$
- ▶ The sender decrements  $CR_{VCi}$  when it transmits a PHIT
- ▶ The sender increments the credit count whenever it receives a credit from the Receiver
- ▶ If  $CR_{VCi} = 0$ , then the sender waits for the credits to get updated
- ▶ The Receiver sends an update only when a PHIT is dequeued from the VC buffer
- ▶ Initial Condition:

- ◆  $CR_{VCi} = m$ , where  $m$  is the number of PHIT buffers configured for  $VC_i$  at the downstream node.

Flow control rules for buffered VC switches with shared and dedicated buffers is as follows:

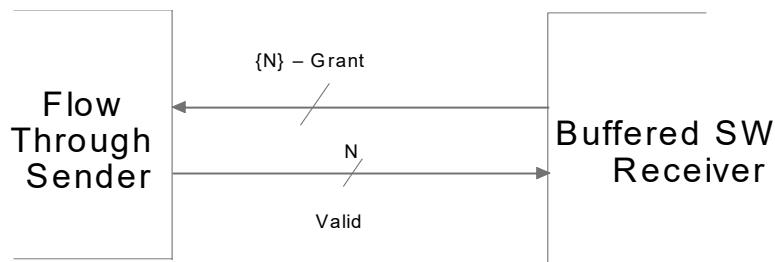
- ▶ Same as above, except the following
  - ◆ The Sender only sends a PHIT if the  $CR_{VCi} > 0 \parallel CR_{Shared} > 0$ .

### 5.6.3 Flow Through and VC Buffer Switch Flow Control

The flow through switches use a Valid-Ready link interface. The Valid and Ready signals are  $N$  bit vectors where  $N$  is the number of VCs supported on the link. There can be only one valid and ready bit asserted on the link.

In the Valid-Ready interface, a PHIT is considered transferred if both the valid and ready signal are asserted in the same cycle.

Since the flow through switch uses a Valid-Ready interface, the buffered switch interface has to be modified from a Valid-Credit interface to a Valid-Ready interface. That is, instead of asserting a credit bit every time a dequeue happens, it will assert a valid signal when a PHIT is transferred from the link into the VC queue. Figure 5-18 on page 76 below illustrates Ready-Valid interface between a Flow-Through switch and the Buffered VC-switch.



**Figure 5-18 Valid -Ready interface between a Flow Through switch and the Buffered Switch**

Below are the flow control rules:

- ▶ Flow Through switch interface is a Valid-Ready interface, i.e., when a valid and ready are asserted on a link during a cycle, the PHIT is considered transferred.
- ▶ The sender asserts a valid signal for the VC chosen by the arbiter to transmit the PHIT.
- ▶ The receiver asserts a Ready signal once the switch determines it can buffer the PHIT. The receiver shall not assert a Ready if the VC queue is full.
- ▶ The receiver does not assert the valid when a dequeue for the VC queue happens, rather it asserts the Ready when the PHIT is buffered.
- ▶ The PHIT is considered transferred when both the Ready and Valid signals are asserted in the same cycle.

### 5.6.4 Routing

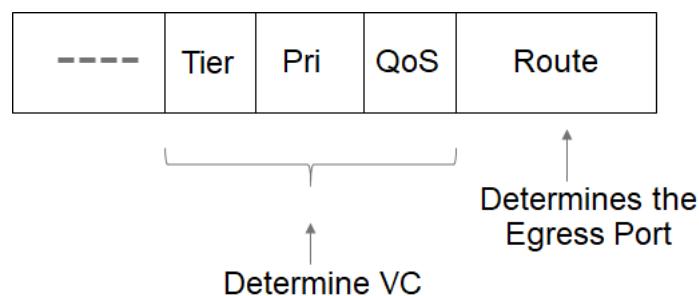
Symphony switches will support both source and incremental routing, however, the first instantiation of Symphony will only support Source Routing. For more information on Routing in Symphony, see section on Routing.

In source routing the routing algorithm determines the complete path between Source  $S_i$  and Destination  $D_j$ . This route is represented in the form of a bit pattern such that each switch uses a certain number of bits in the route field of the packet to determine the desired egress port for the packet. Let those fields be represented by  $R_i$ , where  $i$  is the switch index along the path and  $R$  is the number of bits used by the switch to determine the egress port. The total route from source to destination is expressed as =  $\{R_1, R_2, \dots, R_n\}$  where the route goes through “n” switches.

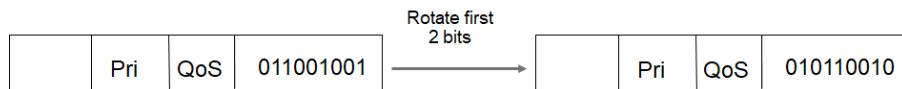
The Route logic will look at the  $R_i$  set of bits for switch  $i$ , decode the bits and rotate them from the front to the back and return the Egress port number.

The route logic if configured, will additionally look at the QoS, or Pri, or Tier bits or a combination thereof and determine the desired Virtual Channel at the egress port. Figure 5-19 on page 77 identifies the packet header fields used for determining the egress port, and the associated VC. Figure 5-20 on page 77, illustrates how the route bits will be rotated once the egress port has been determined.

**Figure 5-19 Packet fields used for determining the egress port and VC**



**Figure 5-20 Route bit rotation at the switch.**



Detailed description of how the fields are mapped and the associated flexibilities is discussed in the Routing section.

The switch route logic can also restrict how the bits are interpreted. For example, a NSEW switch that is configured such that if a packet enters from the eastern port, it can never exit out of the northern port, then route field simply needs a single bit per hop.

The packet format is fixed per Fabric. The number of hops that different packets take from source to destination may not be the same. Therefore some route bits may not be used for some flows and should be set to zero.

## 5.6.5VC Status Fields

## 5.6.6VC Allocation

VC allocation matches valid ingress VCs with output VCs based on the arbitration schemes used by switch.

Let us define the following switch parameters:

- ▶ “N” = Number of Input ports.

- ▶ “M” = Number of output ports.
- ▶ “V” = Number of virtual channels per input port.
- ▶ “W”= Number of virtual channels per output port

---

Note

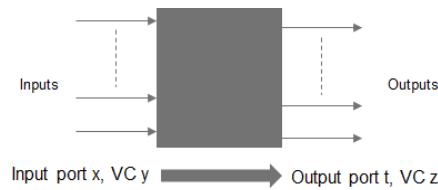
---

The number of input/output VCs per input and output port respectively can be different. The number of input VCs does not have to match the number of output VCs. Also it is quite possible that some input VCs may not be connected to some output VCs depending on how the switch is configured.

VC allocation is done on a packet by packet basis. That is, once the packet from an input VC is allocated an output VC, the allocation holds until the last PHIT of the packet is transmitted. The output VC is marked taken and only released when the whole packet is transmitted from the input to the output.

In general, any packet at any input port and input VC can be mapped to any output port and any VC. We would refer to this as *Any-to-Any* mapping.

VC allocation phase output essentially determines, which input VC is connected to which output VC as shown in Figure 5-21 on page 78 below.



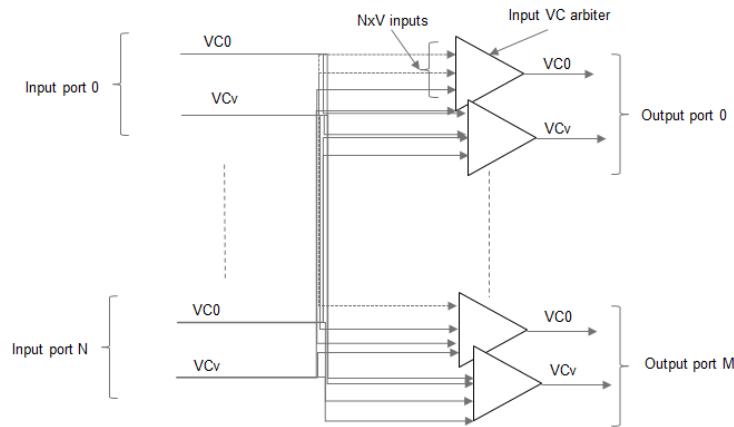
**Figure 5-21 VC allocation phase**

We define the following:

- ▶ VC Status:
  - ◆ Valid: A VC on the input port is considered valid if it has a valid PHIT waiting for transmission.
  - ◆ Busy: An output virtual channel is considered busy if it has already been allocated to an input VC. Once the output VC becomes busy, it will remain so, until the VC sees the last PHIT of the current packet being served.
  - ◆ Idle: A VC is idle if it does not have a valid PHIT. A VC can go from Valid to Idle either because there are no more PHITs in the VC buffer to service or there is a bubble in the packet.
  - ◆ Available: An output VC is considered available if it has not been allocated to any input VC.

Since any a packet in any input VC can request an output VC, the input VC arbiter per output port is an NxV:1 arbiter. The task of the Input VC Arbiter is to select from all the Valid VCs that desire the

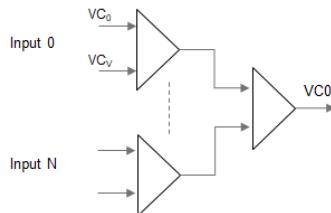
The input VC arbiter can be implemented as a two level hierarchical arbiter, where the first level arbiter is a V:1 and arbitrates between the VCs belonging to the same Input port. The second level arbiter is an N:1 arbiter and arbitrates between VCs belonging to different inputs ports that have a packet that desires to use this output VC. Figure 5-21 on page 78, below illustrates how the arbiters are connected. (Note, this is the control path only)



**Figure 5-22 VC allocation module control path.**

Figure 5-22 on page 79, shows the hierarchical scheduling for NV:1 arbiter.

**Figure 5-23 Hierarchical Input VC arbiter**



The VC allocation module may return more than one allocation VC for the same input port. The Switch allocation then decides based on the available credits and the arbitration scheme which one of the VCs to pick to transmit the PHIT.

Note, that once an output VC has been allocated it is not arbitrated on, until the last PHIT of the packet is transmitted.

---

**Note**

An ideal VC allocation (any input VC to any output VC) module will require  $(NxV) \times (VxM)$  arbitration points which may prove to be expensive. Symphony R1 release will support limited VC mapping, that is, it may not be any-to-any. This reduces the arbitration points to  $NxVxM$ . For a 2x2 switch with 2 VC on each input/output port, the number of arbitration points for the ideal case would be 16 as opposed to 8 for the restricted case. However the number grows if the number of VCs are increased to 4, resulting in 64 arbitration points for the ideal case and 16 for the restricted case.

The VC state is maintained by the switch and comprises of states described above. Every cycle the VC allocation arbitrates to connect input VCs that are valid and output VCs that are not yet busy. The table below shows the state maintained in a 2x2 switch with 2 VCs over a period of 2 cycles. In the first cycle only 3 VCs are valid and are allocated output VCs based on their request as shown in first table below. The status of each output VC is updated from idle to busy. Notice also that input VC<sub>00</sub> and VC<sub>01</sub> are allocated output VC<sub>10</sub> and VC<sub>11</sub> respectively at the same time.

**Figure 5-24 VC status maintained by the Switch (Cycle 1)**

Input VCs	Input VC Status	Output VCs						Avail. Cred
		VC <sub>00</sub>	Avail. Cred	VC <sub>01</sub>	Avail. Cred	VC <sub>10</sub>	Avail. Cred	
VC <sub>00</sub>	Valid					Busy		
VC <sub>01</sub>	Valid			Available		3	Busy	2
VC <sub>10</sub>	Valid	Busy	3					
VC <sub>11</sub>	Idle	Not Avail				Not Avail		Not Avail

In the next cycle, input VC<sub>11</sub> becomes valid. The only available output VC is VC<sub>01</sub>. If the packet in input VC, VC<sub>11</sub> requests output VC<sub>01</sub>, the allocation will be done, otherwise the VC allocation does nothing until one of the other output VCs become available. The table also maintains the number of available credits that the output VC has. Notice that the credit count has changed from cycle 1 to cycle 2 for VC<sub>00</sub> and VC<sub>10</sub> because in this example, the switch transmitted a Flit/PHIT on both output VCs.

**Figure 5-25 VC status maintained by the Switch (Cycle 2)**

Input VCs	Input VC Status	Output VCs						Avail. Cred
		VC <sub>00</sub>	Avail. Cred	VC <sub>01</sub>	Avail. Cred	VC <sub>10</sub>	Avail. Cred	
VC <sub>00</sub>	Valid				Busy			
VC <sub>01</sub>	Valid					2	Busy	2
VC <sub>10</sub>	Valid	Busy	2					
VC <sub>11</sub>	Idle	Not Avail		Busy	3	Not Avail		Not Avail

### 5.6.6.1 Input Port VC Arbiter (Scheduler)

For details on scheduling algorithms see section on Scheduling Algorithms.

The Input Port VC arbiter will follow the rules below:

1. Scheduling algorithms are packet based. That is weight updates and priority decisions are on a packet basis.
2. The arbiter locks once an input is chosen. The lock is released only when the arbiter sees the last PHIT belonging to the packet.

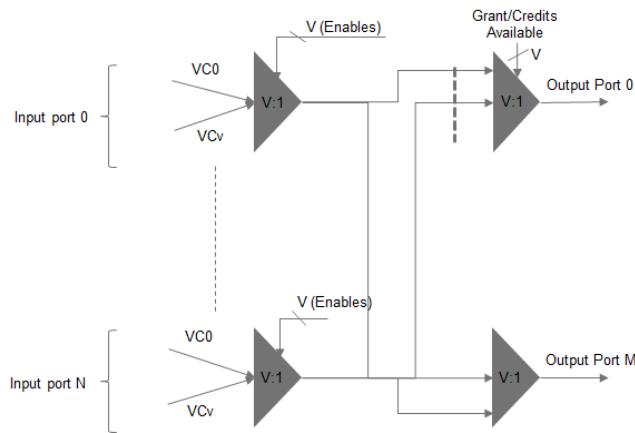
### 5.6.7 Switch Allocation

Switch allocation is done on a PHIT/Flit by PHIT/Flit basis.

The virtual channel allocation phase will result in identifying which valid input port VC is matched with which output port VC. The task of switch allocation is to decide which VC gets to transmit based on whether the VC in question has credits/grant to send the PHIT.

Switch allocation determines which input VC should transmit. Figure 5-26 on page 81, illustrates the switch allocation process for NxM datapath crossbar.

In the previous section, it was mentioned that VC allocation can match one or more VCs from the same input port to different output VCs. The first arbiter chooses the VC based on whether that VC is enabled or not and the scheduling algorithm. The first stage VC is a V:1 arbiter. The second stage arbiter arbitrates between the VCs selected by the first stage. The second stage per output port arbiter selects between the VCs which have available credits/grants. These credits/grants inform the arbiter whether there is buffering available to accept the PHIT/FLIT in downstream VC.



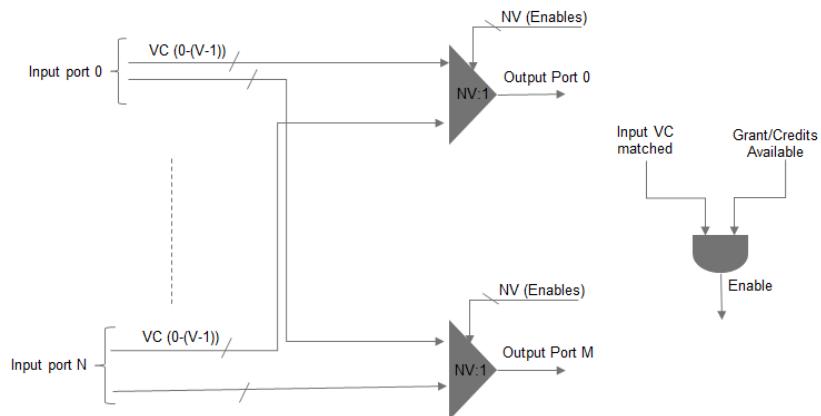
**Figure 5-26 Switch Allocation Control path for NxM Datapath Crossbar**

The enables at first stage arbiter is a V-bit vector which is asserted if the following conditions are true:

1. Input VC is matched (output of VC allocation phase).
2. Enough credits/grant available at the downstream VC to accept the PHIT/Flit.

The scheduler will pick a VC from the valid VCs which has the enable asserted.

If the data path is enabled to transmit multiple PHITs from the same input port simultaneously, then the switch allocation control path would be slightly different and is shown in Figure 5-27 on page 81, below. Note that, in this case then, the data path would be an NVxMW crossbar.



**Figure 5-27 Switch Allocation control path for NVxMW datapath Crossbar.**

### 5.6.7.1 Credits/Grants

The specification uses Credit and Grant interchangeably. A credit of one is equivalent to a grant signal. Using credits provides some relief in timing for the switch allocation arbiters as they do not have to wait for the grant/Ready signal to arrive before the arbiter chooses which VC to transmit the PHIT/FLIT from. The credit update can be done, when the grant or credit signal is asserted without holding up the VC decision.

Setting the number of credits depends on the round trip time between the two switches. If pipeline stages are used on the link, then they also need to be taken into account. The total buffering required will include the pipeline stages along the round trip path.

The credit update is an N field vector. Note that in a switch with NVxMW data path, multiple input VCs on a single input may need to update the credit count for the respective VCs in the upstream switch.

---

————— Note —————  
If there are no timing concerns then, per VC Grant signals can serve as assigning a single credit to the VC credit counter instead of using dedicated credits signals.

---

### 5.6.7.2 Arbiter

Symphony supports multiple arbitration schemes. These include Round Robin (RR), Weighted Round Robin (WRR), and Priority.

It should be noted that a virtual channel is eligible for arbitration at the Master Arbiter, if the following conditions are met:

1. Has a valid PHIT to transmit
2. Credit > 0. That is, the downstream network element has buffer space to accept the PHIT.

The Master Arbiter, picks a VC to transmit based on the scheduling algorithm. When a packet transmission from a VC gets interrupted due to either there is a bubble in the packet and/or the downstream Network element can't be able to accept any more PHITs from the packet, the VC is put on an ordered list of interrupted VCs for that port. The purpose of the ordered list, is enable the arbiter to finish servicing the interrupted VCs in the order they were interrupted. Once the interrupted packet for the VC is serviced fully, the VC is taken off the list.

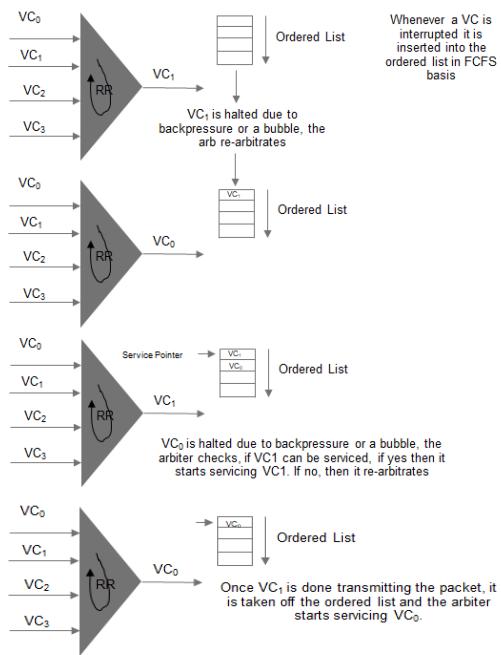
If the interrupted VCs cannot be serviced for some reason, the arbiter re-arbitrates.

---

————— Note —————  
Implementation Note: Alternative to an ordered list, could be that the arbiter services the interrupted VCs by the port number on the arbiter rather than order of interruption. That is, the arbiter simply walks down the port list to check if there are any interrupted VCs. If so, it picks to service the first one it encounters.

---

Figure 5-28 on page 83, illustrates the functioning of the arbiter.

**Figure 5-28 Switch VC Arbiter**

The above arrangement ensures that if packet transmission on a VC gets interrupted because of back pressure, it will be given priority when the next service opportunity arises.

In case of WRR arbitration the arbiter updates weights on a packet basis rather than per PHIT.

## 5.7 Switch Traversal

The switch traversal is achieved by either having an  $N \times M$  crossbar or an  $NV \times MW$  crossbar.

## 5.8 Soft Locking Buffered VC Switch

Soft locking locks are implemented on the Master Arbiter only.

Soft locking for the buffered VC switch follows the following conditions:

- ▶ Condition 1:
  - ◆ Cont. servicing  $VC_i$  in cycle  $t$  iff  $\{Credit > 0 \&& !FirstPit \&& Valid PHIT\}$ .
- ▶ Else Condition 2:
  - ◆ Service the interrupted VCs according to the ordered list described above.
- ▶ Else Arbitrate over all VCs.

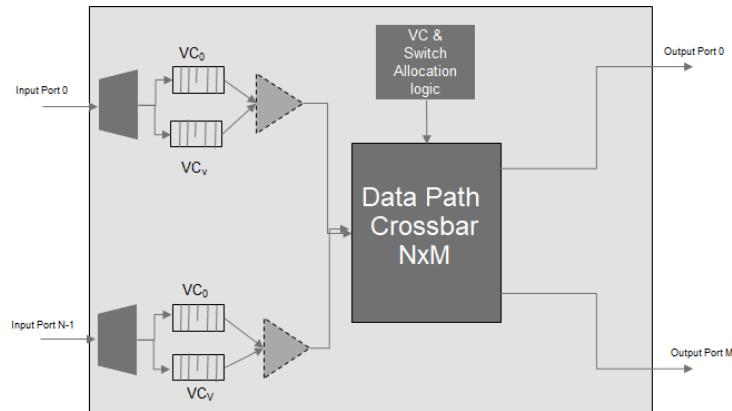
## 5.9 Switch Designs Analysis

The basic switch design is described in the previous sections.

### 5.9.1 Input Buffered Switch

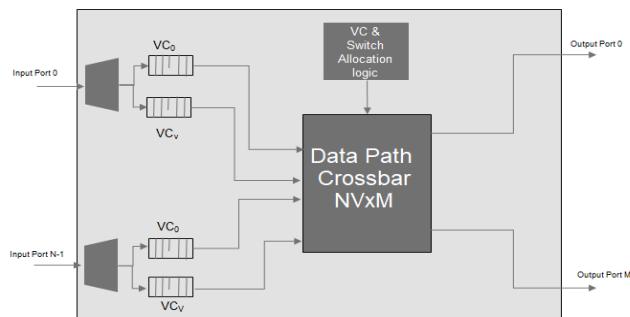
The input buffered switch design is illustrated in the figure below.

**Figure 5-29 Input buffered switch with a NxM data path**



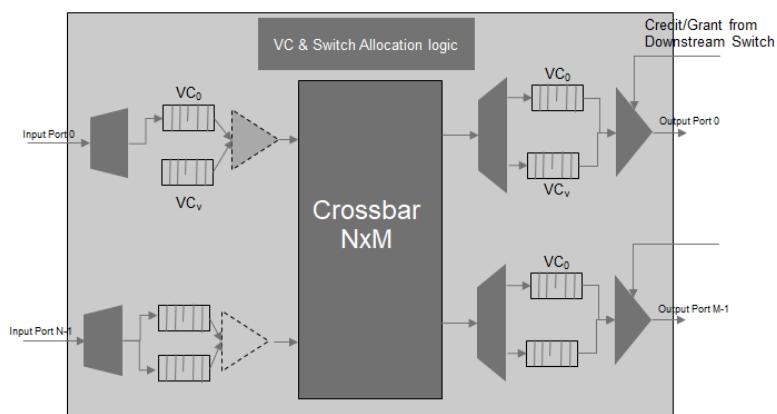
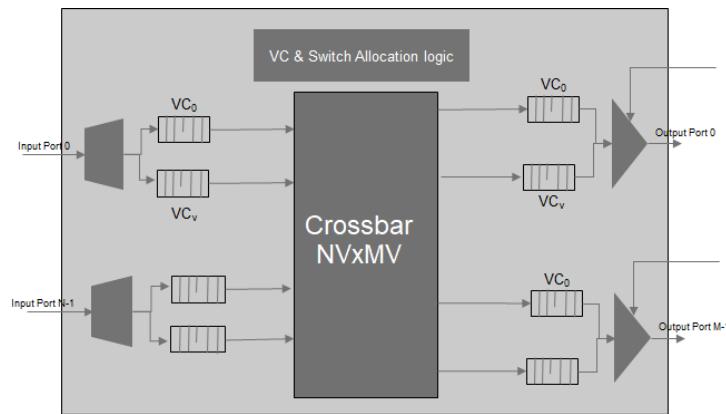
In the input buffered switch, the VC buffers are on the input ports. Figure 5-29 on page 84, illustrates the input buffered switch with an NxM data path. If the datapath is increased to NVxM, the Figure 5-30 on page 84, below illustrates it. With the NVxM data path, there is one less mux on the input that is needed and would result in slightly better performance but at the cost of larger number of crosspoints.

**Figure 5-30 Input buffered switch with an NVxM data path**



### 5.9.2 Input-Output Buffered Switch

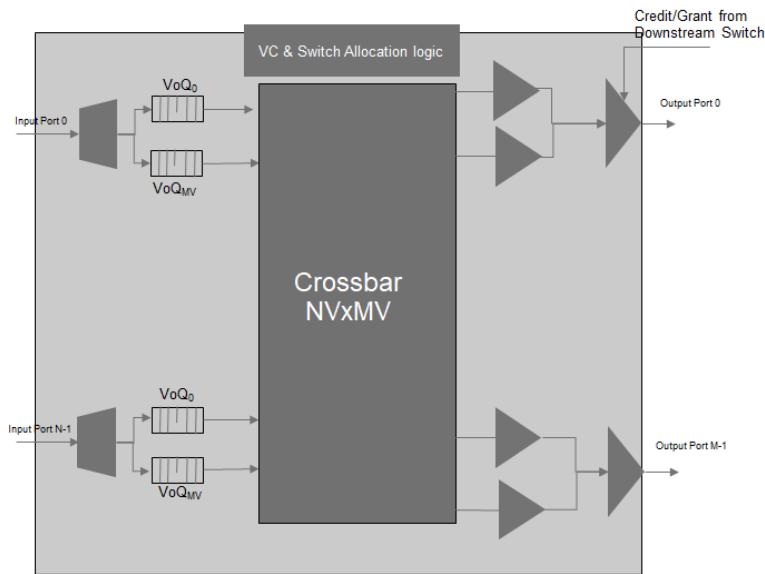
In the input-output buffered switch has VC buffering at the input as well as the output ports of the switch. In Figure 5-31 on page 85, and Figure 5-32 on page 85, a input-output buffered switch with NxM and NVxMW data path are illustrated.

**Figure 5-31 Input-Output Buffered Switch with NxM data path Crossbar****Figure 5-32 Input-Output switch with NVxMW data path crossbar**

The difference between the two flavors is the size/power of the Crossbar and performance improvement.

### 5.9.3 Virtual Output Queue Switch

The VoQ switch has buffering on the input ports. There is buffer for each output port and VC on that port. The number of buffers required for this type of switch is of the order of  $(NxM) \times V$ , hence it very costly. The data path crossbar dimensions are also  $NVxMW$ . However, this switch can achieve 100% throughput in most cases. Below Figure 5-33 on page 86, illustrates a VoQ switch. Notice that each input interface has MW queues.

**Figure 5-33 VoQ switch with NVxMW crossbar switch**

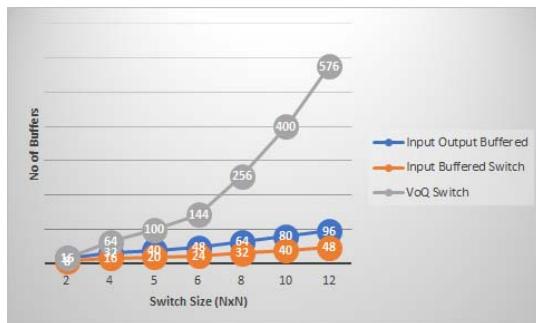
### 5.9.3.1 Analysis

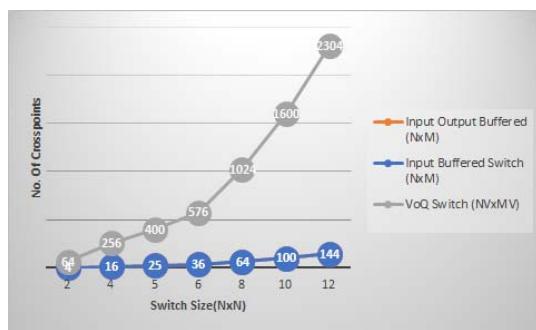
The above three type of switches are evaluated and compared with respect to the following:

- ▶ Amount of buffering required and the size of the crossbar.
- ▶ Performance in terms of Throughput.

The performance results will be published once we have the performance model up and running.

The crossbar and buffer size comparisons are shown in Figure 5-34 on page 86 and Figure 5-35 on page 87 below. The following parameters were used for the comparison: 1) All configurations assume 4 VCs per input port; 2) Buffering requirement is quantized, that is, input VC buffer is taken as one quanta, and output buffer as another quanta. What it means is that the amount of buffering assumed for the input-output buffered switch is twice that of an input buffered switch; 3) The crossbar size is expressed in terms of the number of crosspoints needed and hence the wiring required.

**Figure 5-34 Buffering requirements for different flavors of switches.**

**Figure 5-35 Crosspoint requirement for different flavors of switches.**

Note as shown in Figure 5-34 on page 86, the buffer size in a VoQ switch grows exponentially as the number of inputs grow. Hence for a 5x5 switch the buffering requirement for a VoQ switch is five times that for a 5x5 input buffered switch. Similarly, the number of crosspoints required for a 5x5 VoQ switch are 16 times those required for a 5x5 input buffered switch as shown in Figure 5-35 on page 87.

## 5.10 Starvation Protection

Starvation is defined as the inability of the network and/or a network element to service a port for a long period of time. Note that Starvation is different from deadlock in that a starved port would eventually get serviced whereas a deadlock would leave the network port unserviced for perpetuity.

Starvation is caused when the network and/or the network element is not configured correctly. Usually port starvation occurs in networks that are priority based.

In a priority system, if for example, the rate at which high priority flows inject traffic into the network is not controlled, then there may be scenarios when the high priority traffic consumes all the bandwidth of the link, denying bandwidth to lower priority traffic.

In Symphony, starvation protection at the arbiter level is done by setting up starvation counters for the ports in the arbiter. If the starvation counter for a port counts down to zero, that port is serviced at the highest priority whenever the next opportunity to service appears.

It should be noted, that the function of the Starvation Counter is to only provide an opportunity for the port that is starved to get serviced (wins arbitration at the switch in question) by the arbiter as soon as possible. The function of Starvation Counter does not guarantee forward progress as that depends on circumstances beyond the control of the switch in question.

The following conditions govern the behavior of the Starvation Counter:

1. The Starvation Counter  $SC_i$  for port  $i$  is configured with a count of "n".
2. Every cycle, the port is not serviced (given the port is valid), the  $SC_i = SC_i - 1$ .
3. When  $SC_i = 0$ , the port is serviced at the next service opportunity.
4. When ever port " $i$ " is serviced then  $SC_i = n$ . (Serviced == wins arbitration).
5. For WRR arbitration, once  $SC_i = 0$ , if the port cannot be serviced because  $w_i = 0$ , replenish the weights of the whole arbiter.

If a single counter is used per port, then the Starvation Service opportunity of each VC will be associated with a bit pattern on that counter. When the counter hits that bit pattern, the VC will be serviced at the next opportunity.

In the Flow Through switch, the arbiter will lock once a port wins arbitration. The lock will be released once the whole packet is serviced.

---

————— Note —————

**Implementation Note:** Our first implementation will be based on a single Starvation Counter per port. The counter will decrement every cycle. The VCs on the port will be serviced every time the counter hits a pre-determined bit pattern. That way, the port will be serviced whether it is starved or not, but this arrangement guarantees, there will be no starvation on the port.

---

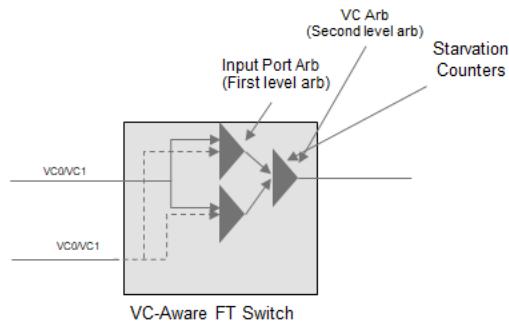
### 5.10.1 Starvation Counters in VC-Aware FT switches

The arbitration for the VC-aware FT Switches is done by a hierarchical two level arbiter. The first level arbiter is used to allocate a VC to an input port for the duration of the packet transfer from that port. The second level arbiter chooses between the different valid VCs, as to which one to allow to transfer the PHIT.

The first level arbiter makes the decision on a packet basis, whereas the second level arbiter makes the decision every cycle.

In VC-aware FT switches the Starvation Counters are placed at the second level arbiter. The First level arbiters are never Priority arbiters (otherwise the switches will deadlock) and therefore an input port will never starve. Once the Starvation counter for a VC reaches a zero count, that VC assume highest priority and is serviced (allowed to make forward progress) at the next available opportunity.

Figure 5-36 on page 88, below shows where the Starvation Counters are placed in the VC-Aware FT switches.

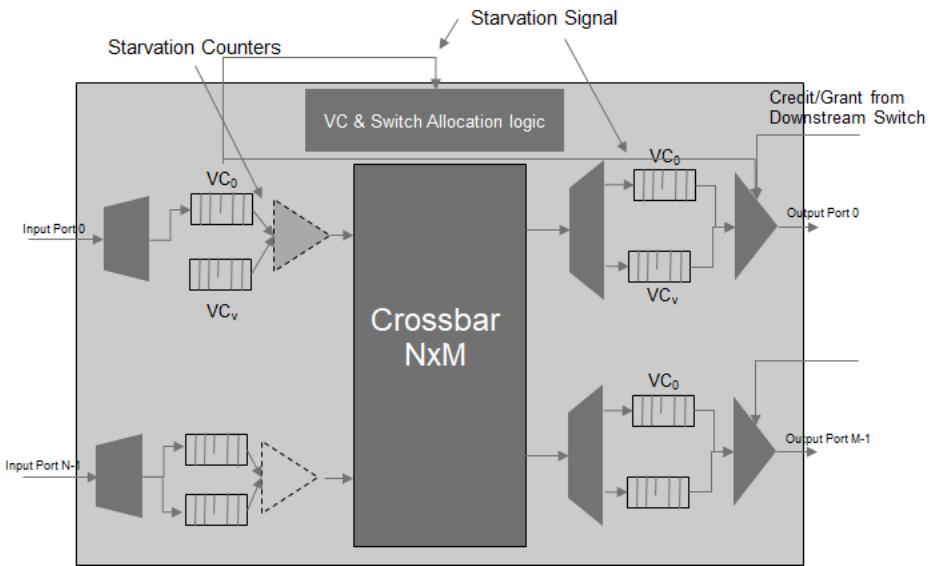


**Figure 5-36 Starvation Counters in VC Aware FT Switches**

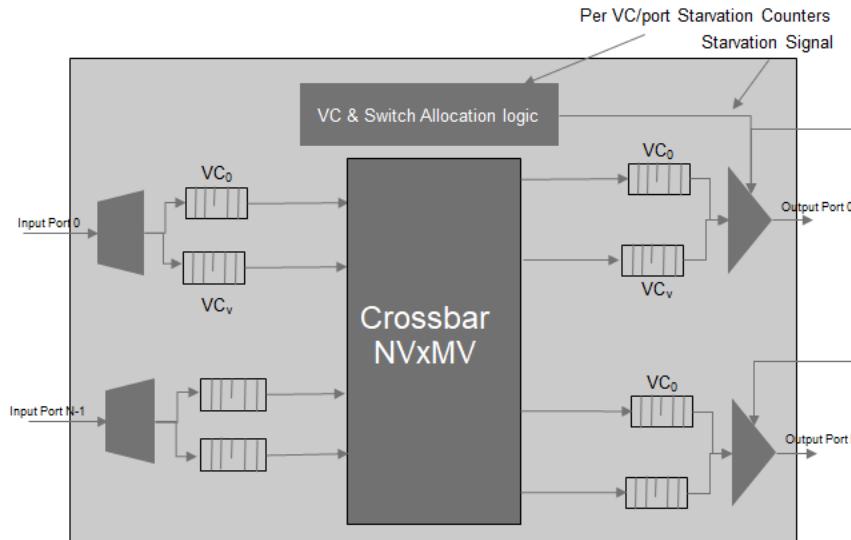
In the VC-Aware flow through switch, a VC that wins arbitration can continue to service the whole packet due to the soft locking mechanism, if no back pressure is exerted by the downstream switches and other network elements.

### 5.10.2 Starvation Counters for VC buffered and Pipelined Switch

In the VC buffered and pipelined switches, unlike the Flow Through switches the control path can be pipelined. As per the description in Section 5.6 “Fully Buffered VC Switch”, the Starvation Counters are placed either at the input port Mux as shown in Figure 5-37 on page 89 or at the VC allocation arbiter as shown in Starvation Counters and Starvation signaling for NVxMW data path switch on page 89. When the starvation counter goes to zero for a VC on the input port, a Starvation signal is asserted for that VC. The starvation signal travels along the path the VC control signal took. That is, raises the priority of the VC in the VC allocation logic as well as raising the priority of the VC in the requested (by the packet) VC arbiter at the output port.



**Figure 5-37 Starvation Counters and Starvation Signaling for  $N \times M$  data path switch**

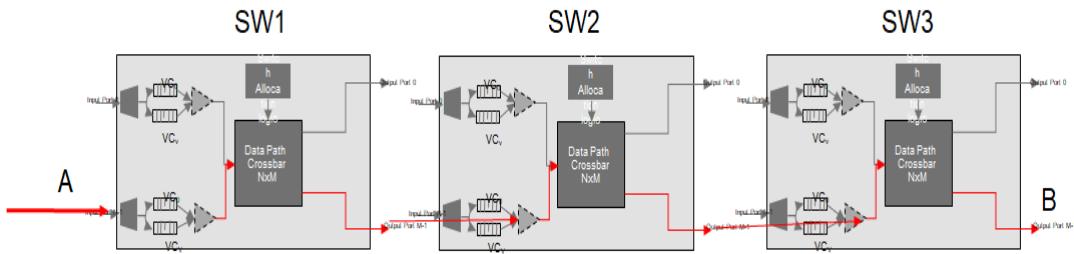


**Figure 5-38 Starvation Counters and Starvation signaling for  $NV \times MW$  data path switch**

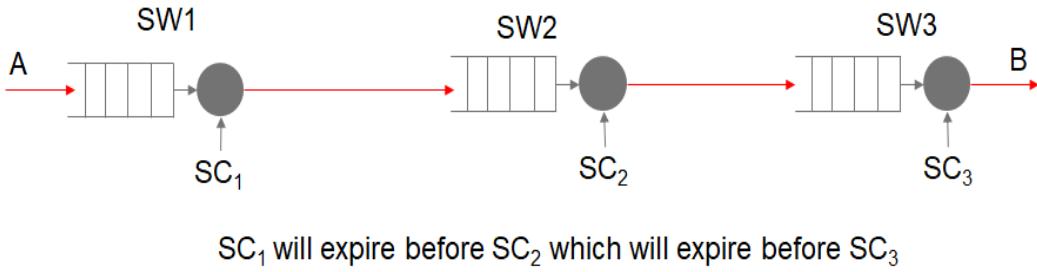
The Starvation Counter is reset once the port transmits the Last PHIT of the packet at the head of the buffer from the VC that raised the Starvation signal.

### 5.10.3 Cascade of Switches

Consider the network shown below in Figure 5-39 on page 90. Flow AB, traverses SW1, SW2 and SW3. This network could be represented using a queue model abstraction as shown in Figure 5-40 on page 90.



**Figure 5-39 Cascade of Buffered Switches**



**Figure 5-40 Queuing model representation of the network in Figure 5-39 on page 90.**

The Starvation Counters for each switch are labeled by the switch number. In a cascaded system if the Starvation Counters are configured identically, then they are expected to expire in the order the flow traverses. That is, SC<sub>1</sub> will expire before SC<sub>2</sub> which will expire before SC<sub>3</sub>, and as such with a very high probability the packets on the port will be able to make forward progress through the network as the upstream switches will already be biased to service the packet when the downstream switch counter expires.

## 5.11 Pressure Signaling

Pressure signaling as the name suggests is a mechanism by which a network element signals to the downstream network elements in the fabric and the interconnect (composed of multiple fabrics) of urgency of packet transfer that is buffered in that particular network element. Pressure is a side band signal that is per port. The pressure signal is a 1 hot low signal whose width is equal to the number of priorities in a fabric with no starvation enabled, or the number of priorities plus one in a fabric where starvation is enabled. The Pressure signal is a *feed forward* signal in that, it travels along or ahead of the packet and raises the priority of the packet that is blocking forward progress, until it reaches a port that is making forward progress.

Pressure can be asserted as result of:

1. Timer expiring, where the timer counts the cycles the packet has waited without getting serviced. This is called starvation.
2. There is danger of an underflow or overflow of a buffer in an Endpoint. For example, an Ethernet controller while receiving Ethernet packets can start running out of buffer space to hold it in coming data. In that case, the Ethernet controller can raise the Panic signal. Other examples could be video buffer underflow/overflow scenario.

The signal conforms to the following protocol:

1. When a packet on a port is starved, pressure is raised to the highest level and “sticks” to the packet. Pressure ahead of the packet is at the next lower level.
2. Pressure travels through the switch from ingress port to egress port when on the previous cycle the ingress port was not granted.
3. Pressure travels across pipeline boundaries if the pipeline stages are stalled.
4. Pressure may hop multiple packets that are blocking the packet caused the pressure until it reaches a port that has a grant signal.

When more than one ingress port raises pressure to an egress port, the highest pressure of all the ingress ports is used on the egress port.

The pressure signal can traverse all the way to the T-ATU. The T-ATU can then forward this signal to the Native protocol unit.



# 6

# Arteris Translation Unit (ATU)

## 6.1 Introduction

This chapter defines the architecture of the Arteris Translation Unit, from here on referred to as the ATU.

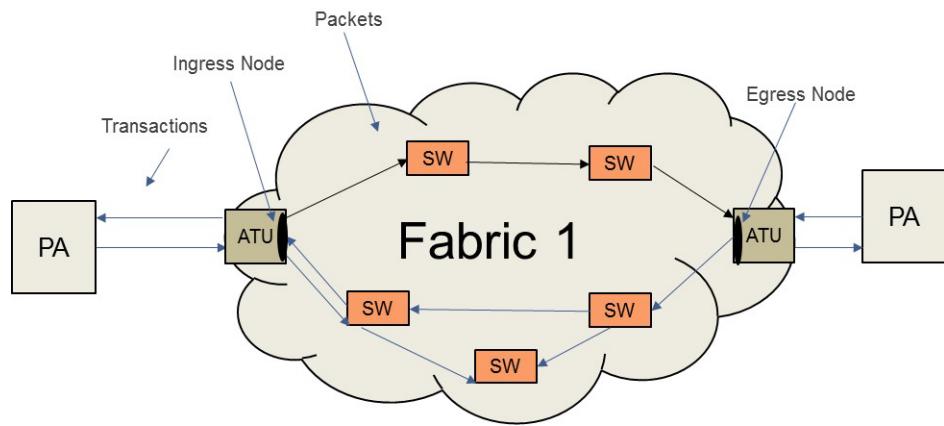
## 6.2 Overall System Architecture

The ATUs connect with the Master/Slave Native Protocol Agents (PA) at the periphery of the Interconnect and symphony interconnect elements via the ATP interface.

The Initiator ATU, converts native protocol transactions into packets and injects them in to the Interconnect. The target ATU, converts these packets back into native protocol transactions and sends them to the Native Protocol Agent.

In special cases, the ATU can interface between two ATP interfaces. These ATUs will henceforth, be called “Adapters”.

**Figure 6-1 Symphony Overall System Architecture**



## 6.3 Arteris Translation Unit (ATU) Overview

The ATU in the Symphony system performs the following functions:

- ▶ Interface with the Master/Slave Protocol Agent
- ▶ Interface with other Symphony components via the ATP interface.

- ▶ Convert the agent side protocol to the Arteris Transport Protocol and vice-versa.
- ▶ Width change adjustment where configured.
- ▶ Determine the intended Target of the transaction received by referring to the Partial Address Map (PAM) in the ATU.
- ▶ Based on the Target ID, add the self-routing bits to the packet header or the destination ID as the case maybe.
- ▶ Manage transaction ordering and other state related to the native protocol.
- ▶ Packetization and de-packetization of the transactions.
- ▶ Mapping of quality of services transactional semantics to link layer semantics.
- ▶ Provide the appropriate level of QoS differentiation based on policies configured by the user.
- ▶ Although not required for each ATU, it will also manage end-to-end credit based flow control based on the traffic management attributes associated with the transaction.
- ▶ When configured, the ATU will perform resiliency checking.

## 6.4 Supported Protocols

ATU will architecturally support the following protocols:

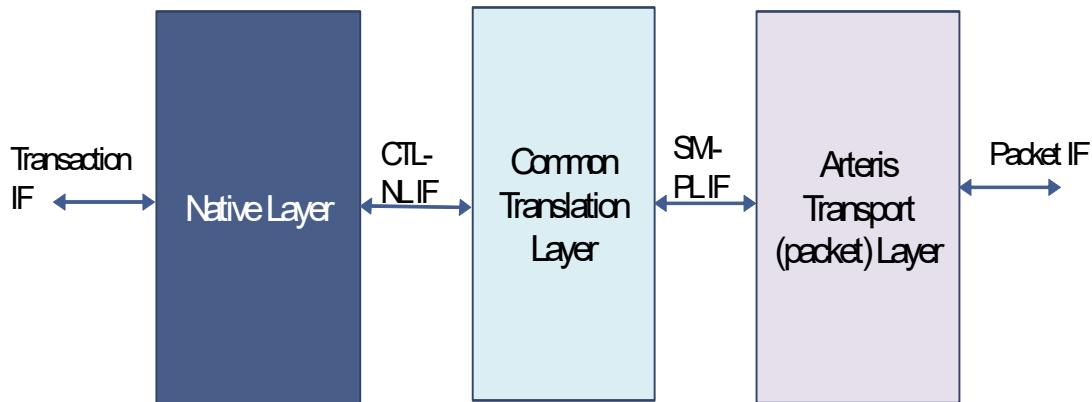
- ▶ AMBA 5/4/3 AXI
- ▶ AXI Streaming
- ▶ OCP
- ▶ APB
- ▶ AHB

ATU will allow interworking between the different protocols.

## 6.5 ATU Processing Layers

The processing in the ATU is logically partitioned in to three layers. These layers are:

- ▶ Native Layer
- ▶ Common Translation Layer
- ▶ Arteris Transport Layer (also referred as - Packet Layer)

**Figure 6-2 ATU processing layers**

The purpose of partitioning the processing into three layers is to provide a standard common layer between the Native and the packet layers. This arrangement allows for different Native Layers to be added to the ATU without changing the common translation layer or the packet layer.

The interfaces between the layers are architected interfaces. The Interface between the Native Layer and the Common Translation Layer is called the CTL-Interface and the Interface between the Common Translation Layer and the Packet Layer is called the Symphony Message Interface (SMI).

## 6.5.1 Native Layer

The Native layer comprehends the semantics of the native protocol (e.g.: AMBA4 AXI, AHB, APB, ..) and translates them in to the semantics of the Common Transfer Layer.

This layer/module's functionality is protocol dependent and will be different for different ATU types. This layer performs the following functions:

- ▶ Interface to the Native protocols (AXI4, APB, AHB, AXI stream, CHI, etc.)
- ▶ Adapts the Interface to the CTL layer.
- ▶ Separate CTL interface to support multiple transaction types. For example the Native-to-CTL interface will support simultaneous READ/WRITE transactions like AXI4. This interface will be parametrizable and will depend on the type of Native interface and the required bandwidth.
- ▶ Multiple native ports having the same Native Protocol (more on this later in the document).
- ▶ Will provide per QoS flow control, if the Native Protocol Master/Slave supports it.

## **6.5.2Common Translation Layer (CTL)**

The CTL interfaces with the native protocol layer, on the Upstream side and to the Arteris Transport Protocol on the Downstream side.

In the special case of the Bidirectional Adapter, the CTL layer will interface with packet transport layer on both, the upstream side and the downstream side.

### **6.5.2.1CTL performs the following functions**

- ▶ Route determination/address lookup based on address Map (per mode/ATU/VN)
- ▶ Support outstanding read/write requests. The number of outstanding requests is parameterized and will be set at configuration time by the user.
- ▶ Support transaction splitting and striping.
- ▶ Input request queue buffering at both input and output ports. Output port buffering is optional. Both the queues are parameterizable. Buffering will be implemented with logically partitioning the shared memory buffer.
- ▶ Provides end-to-end credit checking/maintenance (optional)
- ▶ Rate Limiter per input queue/interface (optional)
- ▶ Virtual network support (optional)
- ▶ Transaction/Request arbiter (more on this later in the document)
- ▶ Transaction support:
  - ◆ Request-Response
  - ◆ Streaming
  - ◆ Others TBD
- ▶ Security
- ▶ Order management.

#### *1.1.1 CTL-Native Interface*

See <https://confluence.arteris.com/pages/viewpage.action?pageId=8520165> for CTL-Native interface details.

### **6.5.2.2CTL-Native Layer side band signals**

Side band signals between the Native layer and the CTL layer are to provide the following:

### **6.5.2.3Credit based flow control support**

CTL layer will provide flow control signals to the native layer. The native layer can merge these signals or keep them separate. The flow control signals correspond to the queues in the CTL layer. These queues are used to buffer transactions separated based on a predefined policy.

The native layer can also back pressure the CTL layer.

Insert picture and talk about it.

### 6.5.2.4 Multiport ATU support

Sideband signal to carry port identification in a multi-ported ATU.

### 6.5.3 Symphony Message Interface

SMI is message passing interface that transfers messages with or without data to and from either the common layer for Symphony or the SMI mux for NCore 3.

- ▶ Flow control for SMI message is based on valid-ready handshake. If both valid (by master) and ready (by slave) are asserted the transfer takes place. The ready may or may not be asserted in absence of valid.
- ▶ There is a separate flow control for the SMI data payload interface (smi\_dp\_\*). The flow control for data payload is independent from the SMI message flow control though there will be some implementation dependencies (like – SMI packetizer will not accept SMI data payload transfer without the corresponding SMI message transfer).
- ▶ Flow control can be per VC (valid-ready per VC), with only 1 VC transfer accepted per cycle (only 1 ready asserted). The VC selection can change per cycle.
- ▶ Messages per flow/VC are not interleaved. The next message does not start before the current message finishes.
- ▶ SMI can be with or without data payload interface (parameterized). Even with SMI data payload interface, a given message can have no data payload (dp\_present set to 0 in SMI message).
- ▶ SMI message can include non-data payload, which is handled like just another message field
- ▶ All SMI fields are parameterized, and can be optimized out based on configuration
- ▶ The SMI message length is an optional field for legato/NCore but a required field for presto (for read request messages). The data payload uses last indicator to indicate the end of the data and thus makes length field optional for those messages (can be used for length error check).
- ▶ Route can be generated by path lookup using {target ID, steer}, and so either route or steer (with target ID) is defined for a given interface.

For SMI message interface signals and their description, see <https://confluence.arteris.com/pages/view-page.action?pageId=8520165>.

### **6.5.3.1 Flow control support**

Backpressure signal per output queue from native layer. Native layer interprets the flow control <-> queue mapping.

## **6.6 Arteris Transport (Packet) Layer (ATL/PL)**

The ATL layer extracts the information from CTL to create a transport packet. ATL includes the following:

- ▶ Packetizes messages passed down from CTL.
- ▶ Depacketize packets coming in to the ATU from the link (fabric) into messages.
- ▶ De-packetizes the requests/responses coming from the transport fabric
- ▶ Handles Link-to-link (switch) back pressure
- ▶ Packet consists of one or more PHITs, and contains the following sections:
  - ◆ Data Link layer: The header PHIT contains the Data Link layer information which is used by the elements in the Interconnect to make data link decisions on the packet.
  - ◆ Network layer: Contains the information that CTL needs for processing packets/messages.
  - ◆ Transaction layer: Contains information needed by the Native layer.
- ▶ Payload Data and meta-data
- ▶ Data width adaptation (packing/unpacking of data into packets).
- ▶ Handle clock crossing for different interfaces.

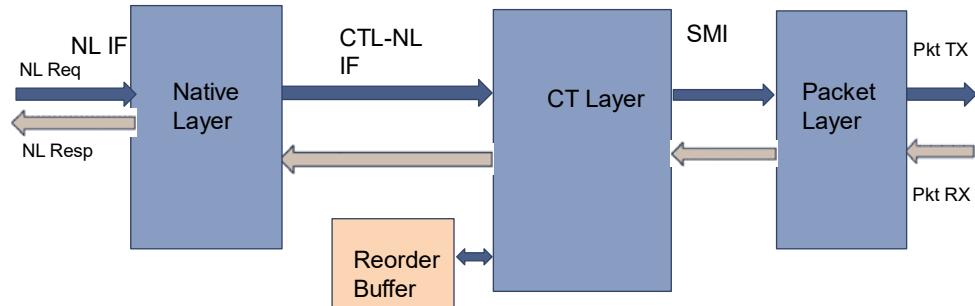
Refer to the packet protocol spec for details on the packet format.

## **6.7 ATU Types**

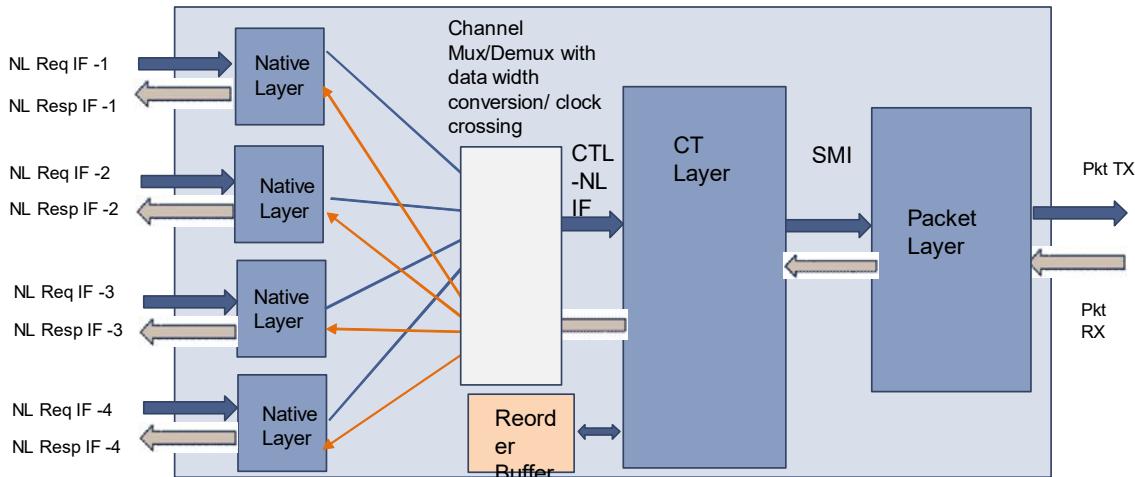
Multiple types of ATUs will be supported

### **6.7.1 Single Ported ATU**

This is the regular ATU – one Native, CT and ATP/packet layers.

**Figure 6-3 Single-ported ATU.**

## 6.7.2 Multi-ported ATU

**Figure 6-4 Multi-ported ATU with common native layer**

The figures above show the multi-ported native layer ATU. All the native ports are of the same protocol type. There is an additional channel mux/demux functionality added to connect the multiple native layer modules to a single CT layer module.

The CT Layer and packet layer remain unchanged. Both layers are shared between the different native layer interfaces. A message from the different interfaces will not be interleaved on to a single packet, that is, no message packing is done. For target ATU, there will be a slave address map used to decode the slave address and map to one of the native ports. For further optimization, a single native layer can interface to multiple native ports (for simple protocols like APB - by decoding separate chip select per APB slave).

- ▶ One native layer with mux/arbitrer on the initiator side, and demux on the target side

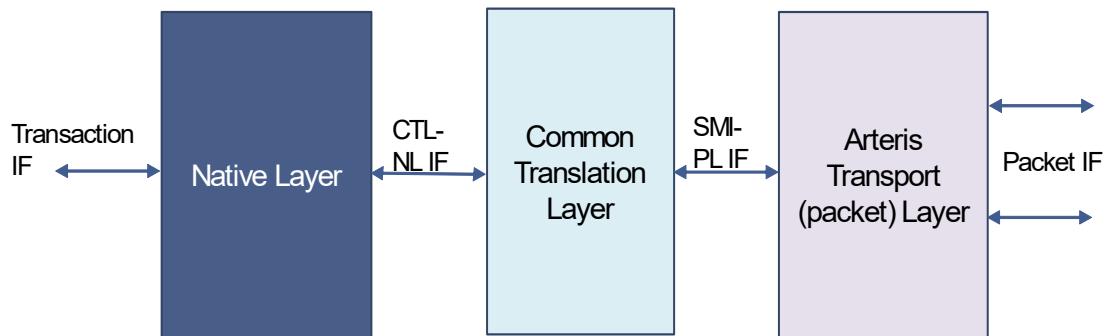
- ▶ All interfaces should be of same native protocol (no protocol muxing) and same clock domain.
- ▶ Support simple non-bursting protocol like APB (AHB, AXI support is being investigated for efficiency and added complexity)
- ▶ There will be a simple round robin arbiter controlling the mux select. Only one interface will be active, others will be back-pressured (ready de-asserted).
- ▶ Support slave address map on the target side for de-muxing to multiple external slaves

### 6.7.3 Multi-ported ATP ATU

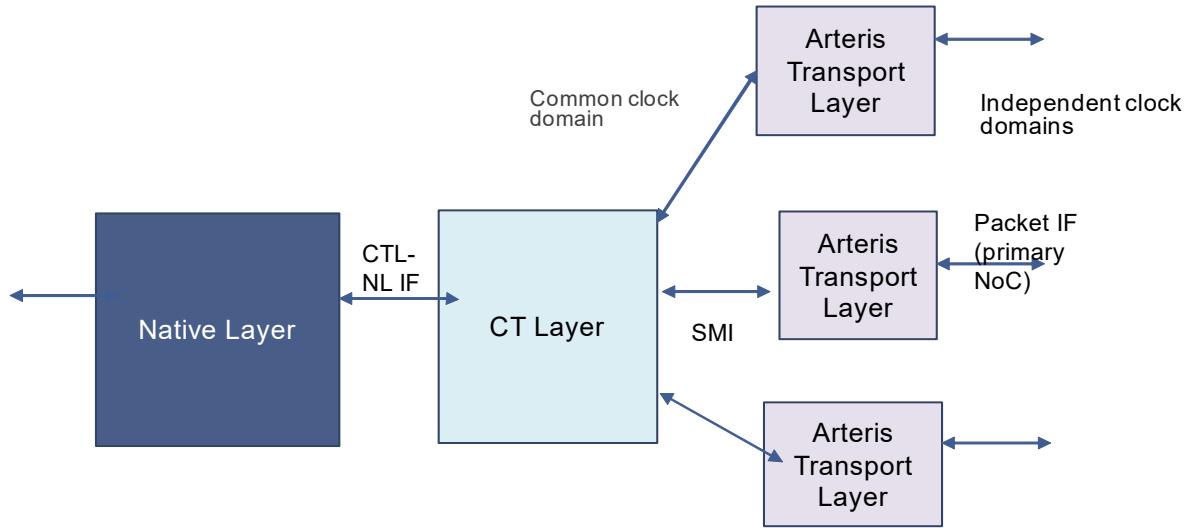
A single ATU can have multiple ATP ports to interface to –

- ▶ separate switches in the same NoC (like separating high priority/low priority traffic): This case can be supported by same ATP layer having multiple ATP ports (port lookup is done as part of route lookup).

**Figure 6-5 Multi-ported ATP**



- ▶ separate NoCs – support additional special NoCs {Configuration NoC, Control NoC, Trace/Debug NoC}: This case can be supported by same ATP layer having multiple ATP ports (single ATP layer has to handle different packet formats) or by using multiple ATP layers with single port each (separate out the packets per NoC, and prevent traffic on one NoC affecting another).

**Figure 6-6 Multi-ported ATP for multiple NoCs**

The figure above shows a 3-NoC ATU (1-primary NoC, 2-additional NoCs). The CT layer generates and consumes messages to/from the additional NoCs (no traffic to/from Native layer) while the native layer sends/receives transaction messages targeted for primary NoC only.

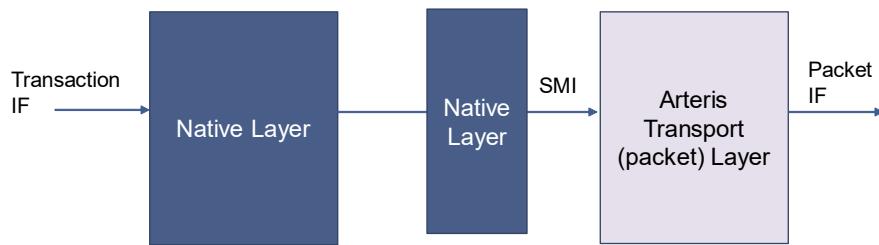
### 6.7.4 Streaming ATU

The streaming ATU is an optimized version of the standard ATU with minimal CT layer. It does native protocol adaptation and packetization. This ATU supports minimal features. There is no reordering, arbitration, etc. It is also unidirectional, i.e., it does not support any context for responses.

Use case for the Streaming ATU are:

- ▶ For point-to-point AXI streaming
- ▶ Switched AXI streaming. Switching will be based on destination ID.
- ▶ Trace and debug data transport.

Streaming ATU is shown below in figure below -

**Figure 6-7 Streaming ATU**

### 6.7.4.1 Native Layer

The native layer interfaces to the Native Protocol (e.g.: AXI Streaming). It also performs the stream splitting to conform to the max packet size. There are no responses. No reordering is done in the network.

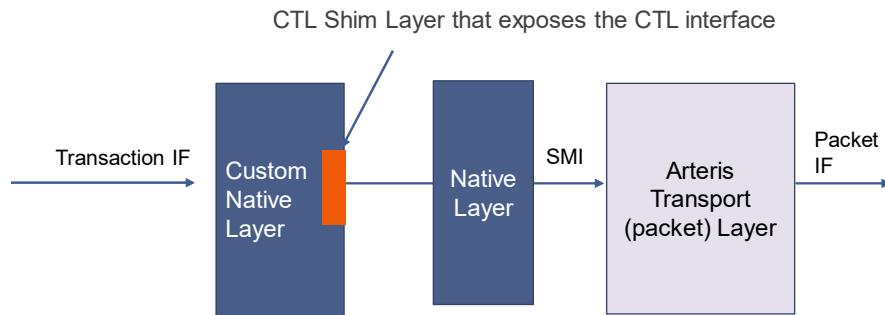
### 6.7.4.2 Packet layer

The packet layer has minimal – link/network layer header, and primarily packetizes the payload (+side-band data).

The point-to-point link can have pipeline stages, clock crossing buffers etc. to meet timing and perform any data-width conversion or clock crossing.

## 6.8 Custom Native Layer ATU

A custom native layer can be supported by replacing the Native layer in the ATU with a thin shim layer obfuscating the CTL layer-to-native layer interface. The shim layer will hide any signal/functionality that need not be exposed, and only expose signals that are deemed necessary and safe. Customer can design a widget to interface their Protocol Agent to this native layer-shim (Interface will be like that of CTL, that passes messages – the details are TBD).

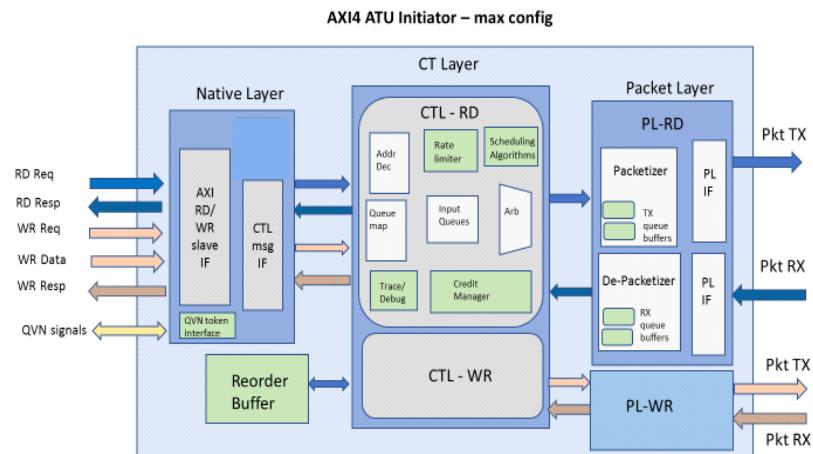
**Figure 6-8 ATU with Custom Native Layer Support**

## 6.9 ATU Configuration

AXI4 ATU is described as an example ATU here. The only module that would change between different ATU types is the native layer. Details of each of the sub-modules shown in the diagrams below will be discussed later in the document.

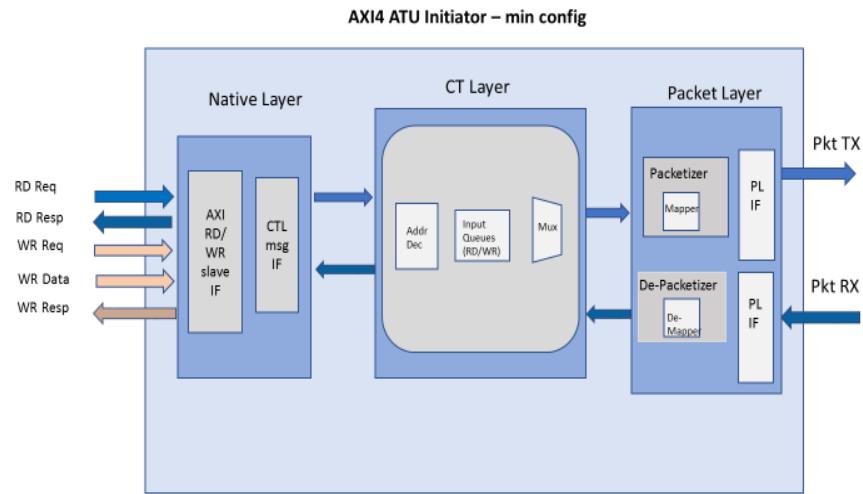
ATU top-level diagrams are show below:

## 6.10 AXI4 Initiator ATU (max config)

**Figure 6-9 Maximum Configuration AXI4 Initiator ATU**

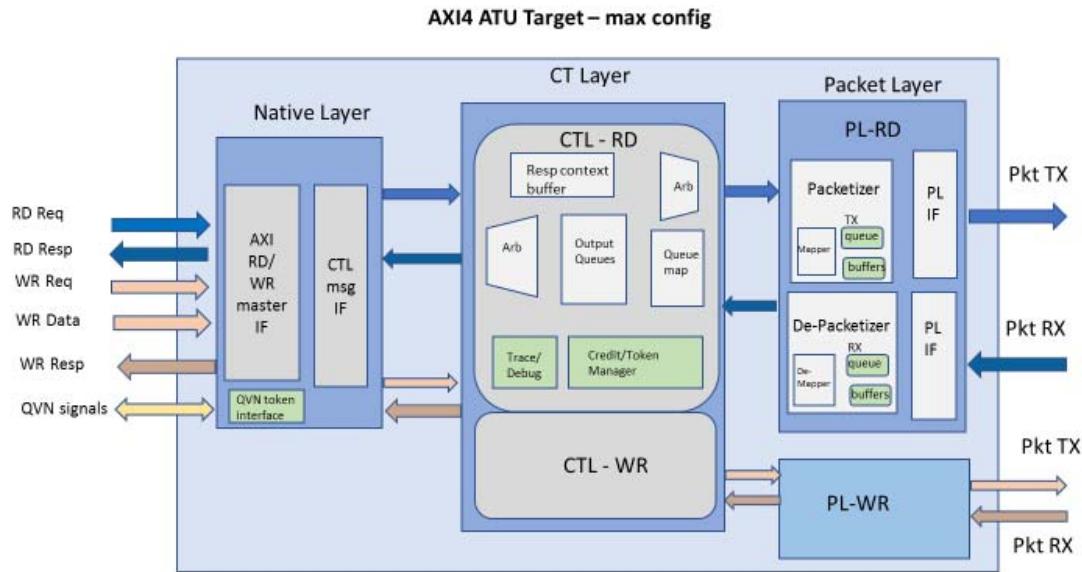
## 6.11 AXI4 Initiator ATU (min config)

Figure 6-10 Minimum Configuration AXI4 Initiator ATU



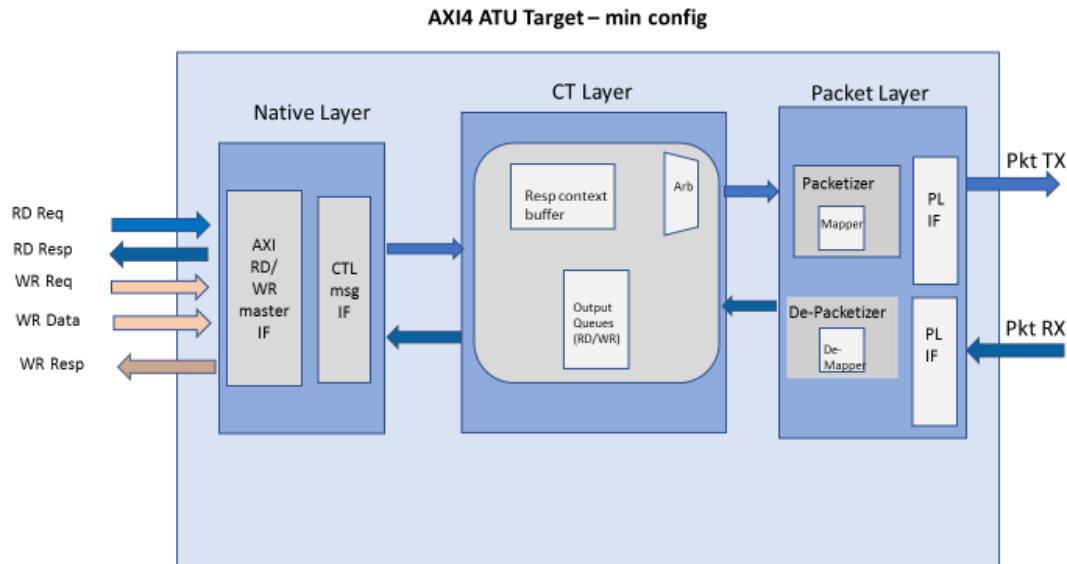
## 6.12 AXI4 Target ATU (max config)

Figure 6-11 Maximum Configuration AXI Target ATU.



## 6.13 AXI4 Target ATU (min config)

Figure 6-12 Minimum Configuration AXI4 Target ATU.



## 6.14 AXI4 ATU

### 6.14.1 Native Layer for AXI4

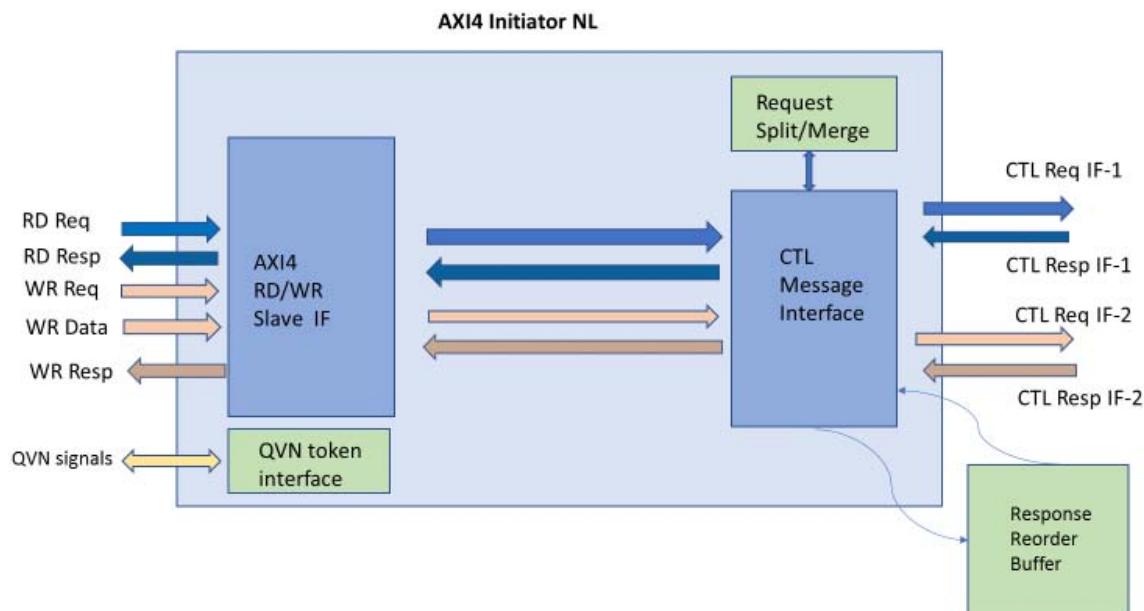
The native layer block provides the interface to the native protocol and adapts it to the CTL. The main sub-blocks of native layer (for AXI4) are –

- ▶ AXI4 Master (target) or Slave (initiator) interface
- ▶ CTL interface (with protocol adapter)

### 6.14.2 AXI4 Slave Interface

This module implements the AXI slave interface logic to interface to the external AXI master.

There are total of 5 channels on the AXI interface (read request/response, write request/data/response channels). These are mapped to the corresponding CTL interfaces, which can be shared or separate for reads and writes (one or two CTL request/response interfaces).

**Figure 6-13 AXI4 Initiator Native Layer.**

If only request or write channels are used on native interface, then only 1 CTL interface will be implemented.

### 6.14.3 CTL message interface

The CTL message interface module maps the AXI requests to CTL messages (read/write) per CTL interface (as described earlier). Similarly, the responses from CTL interface are mapped to AXI responses.

### 6.14.4 AXI4 Native Layer for the Target ATU

The Native layer for Target ATU is similar to the Initiator ATU as described above. The CTL message interface, includes protocol adapter, to convert the CTL message into the AXI4 transaction..

### 6.14.5 Common Translation Layer (CTL)

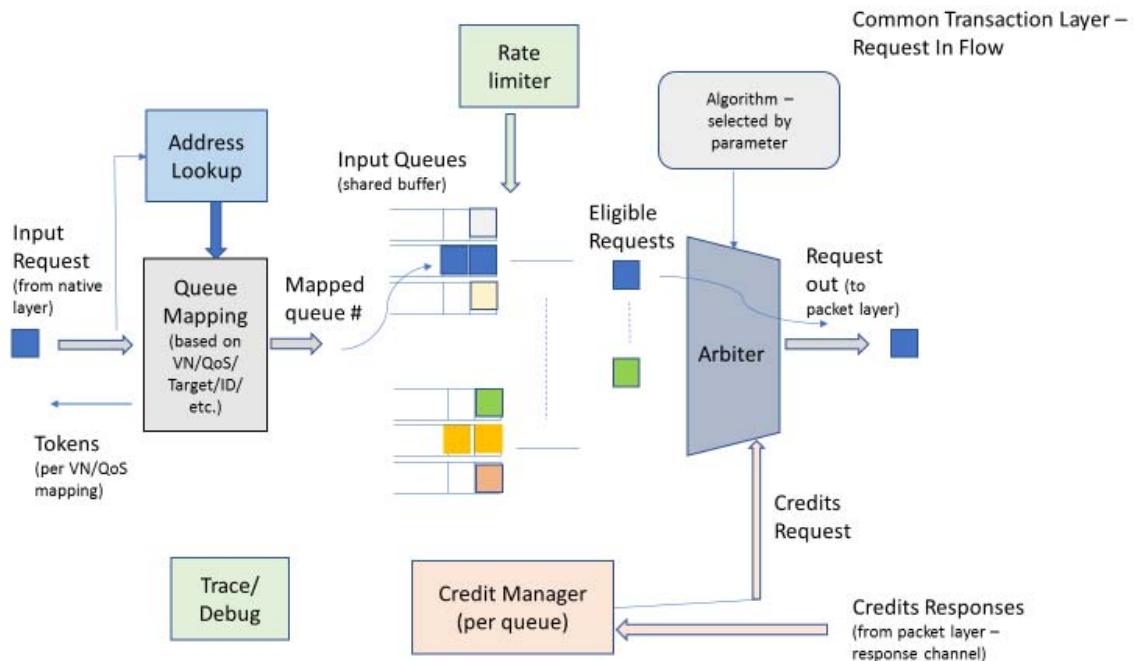
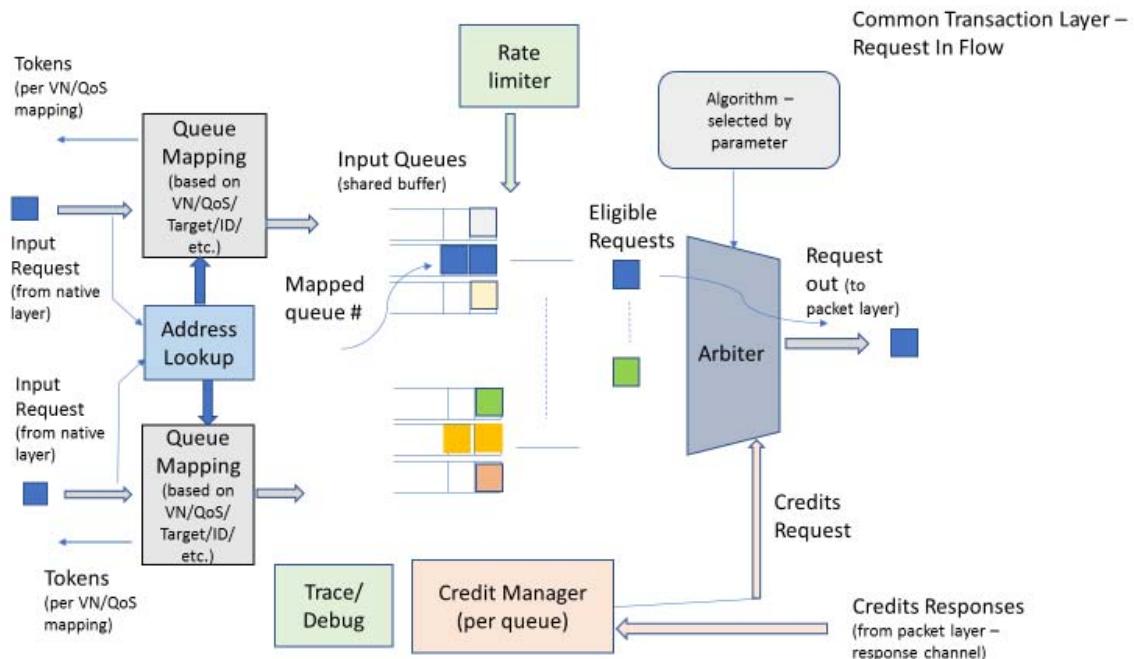
The common translation layer is independent of the native layer and will have some functionality added/removed based on the ATU parameters.

## 6.15 CTL Initiator block

The CTL initiator block receives the requests on CTL interface from NL and sends the requests to packet layer on the extended CTL interface, and vice-versa for the responses.

There are two versions of the CTL initiator block –

- ▶ One input CTL interface, one output CTL interface
- ▶ Two input CTL interfaces (read and write), one output CTL interface

**Figure 6-14 1:1 CTL interface.****Figure 6-15 2:1 CTL Interface.**

The main sub-blocks of CTL module are –

## 6.15.1 Address Lookup

The address lookup uses the Partial Address Map (PAM) generated by the SW based on the user configuration of the initiator ATU or derived from the System Address Map (SAM). The generated values (hard-wired defines/parameters) can be used directly or as default values for the configuration registers, representing the PAM. The address bits and ranges in PAM can be subset of those in the SAM. The PAM table entries consist of –

- ▶ Base address and size: The size must be power of 2, and the base address should be aligned to the size of the address range.
- ▶ Target ID: The assigned target\_ID (by SW) for the address range
- ▶ Striped: Index to stripe group table if the address range is striped. The stripe group table consists of details to handle the address striping – the number of striped targets, the respective target\_IDs and routes, striping function select and stripe size.
- ▶ Route: Routing bits for the route to get to the target (determined by SW)

The figure below shows an example PAM –

**Table 6-1 Partial Address Map Table.**

Base Address	Size (power of 2)	Target_ID	Route	Striped (optional)	Stripe Group (optional)
0x0000_8000	0x4000	1	12'h011	0	X
0x0001_0000	0x1_0000	4	12'h020	0	X
0x0002_0000	0x1_0000	2	12'h032	0	X
0x4000_0000	0x4000_0000	X	X	1	0
0x8000_0000	0x4000_0000	X	X	1	1

**Table 6-2 Stripe Group Table.**

Stripe Group	Stripe Target count	Target IDs	Route	Stripe_func_sel	Stripe_size (power of 2)
0	2	{10,11}	{12'h040,12'041}	0	0x400
1	3	{12,13,14}	{12'h042,12'h043,12'h044}	1	0x100

Route information –

The routing information for a given target can be fixed (part of the generated PAM) or configurable – stored in configuration registers, which feed into PAM table and Stripe group table. If the routing entries

are stored in configuration register, then the SW determined route values are the default values for those configuration registers. The route lookup gets done as part of the target ID lookup.

#### Memory map modes –

There can be separate memory map (PAM) defined per mode, like – {boot mode, secure mode, normal mode, any user defined mode}. The mode selection will be done using select input(s) into the Fabric or via user defined bits.

#### Striping –

An address range in the PAM table that is marked as striped, can stripe the address across multiple targets, and the striping group provides the information to handle the striping-

- ▶ Stripe Count – number of targets striped
- ▶ Stripe size
- ▶ Stripe function -
- ▶ User-defined striping bit(s): select one or more bits as target selector bits (power of 2 targets)
- ▶ Customized striping: A drop-in customized/library module that determines the target (not limited to power of 2 targets).

A request spanning across striping boundary, will be split and the responses have to be re-ordered or interlocked.

#### Mapping –

The striped addresses to a target are not contiguous and can use address mapping to make it look contiguous/ fill up the address gaps. The address mapping feature is optional, and can follow the following mapping options (parametrized) –

- ▶ Mapping for striped bits - Shift right the bits above the striping bits by number of striping bits, or move the MSB bits to replace the bits used for striping.
- ▶ Customized Mapping- A drop-in customized module that determines the address given the target selection (It can be part of the customized striping module or separate).

#### Target Aware Request Split:

There can be optional columns in the PAM for indicating target aware request splitting. There are parameters to enable request splitting per target for –

1. Boundary based (like AHB – 1KB boundary)
2. Size based (limit to max burst that target can accept)
3. Protocol incompatibility (like split if target does not support wrap or has smaller burst support)

For more details on Partial Address Map, please see the section on Memory Map.

## **6.15.2 Queue mapping –**

It maps an incoming request to one of the input queues. The mapping function is configurable/parametrized function of (VN, QoS, target ID, target address)

### 6.15.3 Input Queues –

Optional buffer space for storing requests – read/write requests, and write data. The Input queue buffering will be shallow and would backpressure the native layer when full.

### 6.15.4 Credit Manager (optional) –

The credits are described in detailed in the credits mechanism section. The credits can be threshold/priority based per queue or ATU. The credits can be dedicated or shared. The credit manager allows the requests that have credits to be eligible for arbitration and deducts the credits used once a request gets sent out to packet layer.

### 6.15.5 Rate Limiter (optional) –

#### 6.15.5.1 Single Leak Bucket

Rate limiter tracks and limits the rate of request injection by that ATU.

- ▶ Rate limit per request type - read/write per QoS/Virtual network/CTL queue.
- ▶ A given request type is allocated a bucket size and refresh interval (bucket will be replenished at every refresh interval)
- ▶ Bucket of size N can transfer  $N \times (4B/16B/64B/256B)$  - configurable
- ▶ Bucket size is normalized to start at:  $N + \Delta$  ( $\Delta$  - configurable per bucket) so that a bucket size does not go negative when the transfer request starts with bucket size >  $\Delta$ .
- ▶ For every request, the bucket size is decremented by the request size
- ▶ Bucket is considered empty (for new request) if bucket size  $\leq \Delta$
- ▶ Refresh interval (T: 1 to 1023) for a bucket can be in 1K/1M clock units. At every refresh interval, size N is added to the bucket (bucket size capped at  $N + \Delta$ ).

#### 6.15.5.2 Dual Leaky Bucket

The dual leaky bucket Rate limiter is to allow Masters to send traffic over their “Sustained Rate”. This overage will be referred to as the “Excess Rate”. The “Excess Rate” traffic will be transmitted at a lower priority than the “Sustained Rate”.

The dual leaky bucket has two buckets of size “N” and “M” and two refresh rates to match the sustained and excess rates. The algorithm is as follows:

1. Parameters:
  - a. Sustained Rate Refresh rate:

## **6.15.6 Arbiter (optional) -**

The arbiter selects one from the available eligible requests to be forwarded to the packet layer based on the arbitration policy selected (parameterized or configured) -

- ▶ Round robin
- ▶ Priority with starvation avoidance
- ▶ Deficit Round robin

## **6.15.7 Trace and Debug (optional) -**

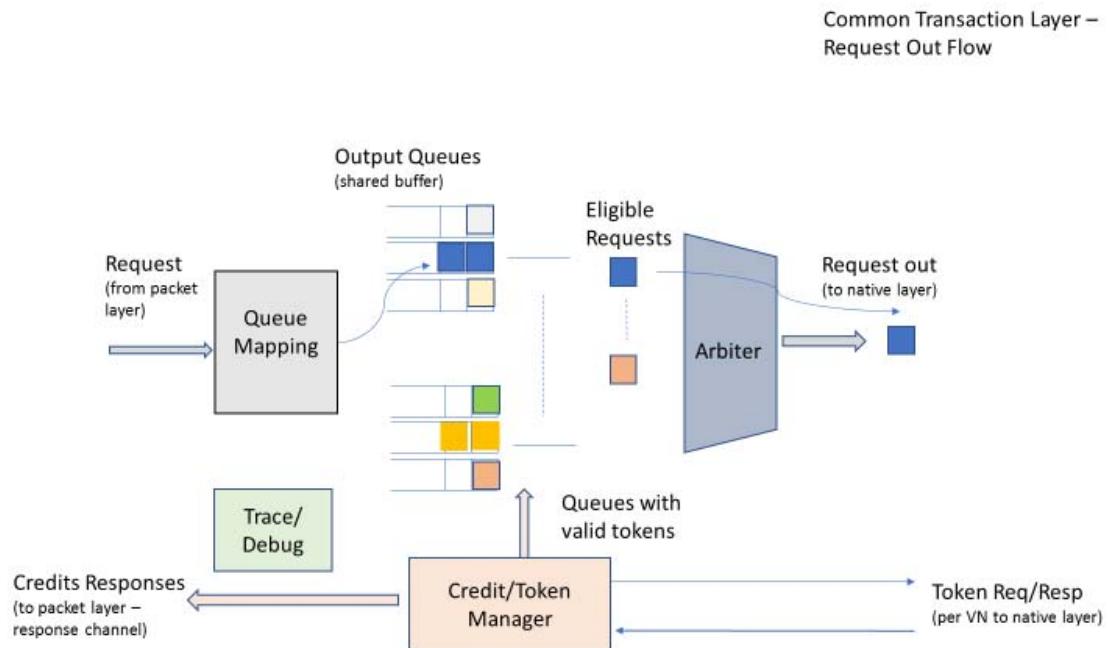
This module is for capturing trace and debug data for the ATU. The details are TBD.

## **6.15.8 Response flow**

The response flow through CTL source block is just flow through and updates the outstanding requests context.

## **6.16 CTL Target block**

The CTL target block receives the requests from packet layer on the extended CTL interface, and sends the requests to Native layer on CTL interface, and vice-versa for the responses.

**Figure 6-16 CTL Target Block.**

The main sub-blocks of CTL module are –

### 6.16.1 Queue mapping (optional)

It maps an incoming request (from the packet layer) to one of the output queues. The mapping function is configurable/parametrized function of (VN, QoS, target address). In the minimal configuration, there would be no queue map and transactions would be queued based on transaction types.

### 6.16.2 Output Queues (optional)

The output queues are corresponding to the input queues of the source ATUs. The input request is mapped to the queues as configurable/parametrized function of (VN, QoS, ID, target address). The queues are implemented as linked list of buffers allocated by the credit manager which manages the shared buffer.

The queues can have fixed allocated buffer space or dynamically allocated from a shared buffer pool (to support shared credits and speculative credit domain). There is a buffer manager that allocates and manages buffer space for dynamic allocation – allocated upon a credit request or upon receiving a speculative request, and deallocated when the request is de-queued.

### 6.16.3 Credit and Token manager (optional) –

The credits are described in detailed in the credits mechanism section. The credits can be dedicated or shared, the dedicated credits are allocated per initiator queue/VN. The credits are released back (if dedicated) once a request is sent out from the output queues by the arbiter (towards the native layer).

The token manager requests tokens for the requests in the output queues, corresponding to the mapped VN, to make the request eligible for arbitration.

## 6.16.4 Arbiter (optional) -

The arbiter selects one from the available eligible requests to be forwarded to the packet layer based on the arbitration policy selected (parameterized or configured), which are subset of the arbitration algorithms supported in the source CTL module.

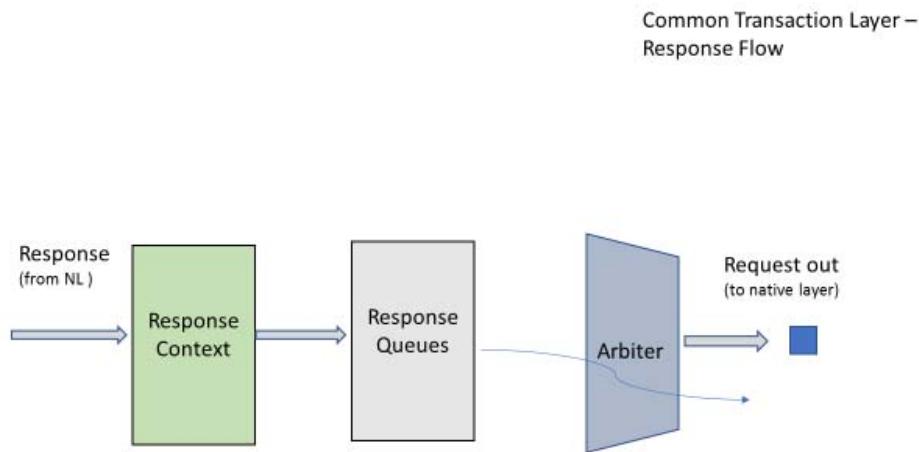
- ▶ Round robin
- ▶ Priority with starvation avoidance
- ▶ Deficit Round robin

## 6.16.5 Trace and Debug (optional)

This module is for capturing trace and debug data for the ATU. The details are TBD.

## 6.16.6 Response Flow

*Figure 6-17 Response Flow.*



### 6.16.6.1 Response (Request) Context

The response context is a data storage that holds the context data for each outstanding request to generate the corresponding CTL response message and enables request ID remapping. It is used to track the request IDs in use (input and mapped request IDs) and remapping of ID is done while ensuring ordering for same IDs from same source. The context data (VC, priority, etc.) is used to map the response message

to one of the response queues. Also, the source ID is used to lookup the routing information for the responses.

ID compression - The number of outstanding requests can be greater than the number of unique requests for the given slave ID width. The mapped IDs are assigned honoring the ID ordering rules (and unifying the mapped IDs between different masters). When there are no free IDs to use, more than one request (with different request/source ID) can get assigned the same mapped ID (ID in the request going to the external slave).

### 6.16.6.2 Response Queues

There can be one or more response queues that holds the response messages till they get selected by the arbiter.

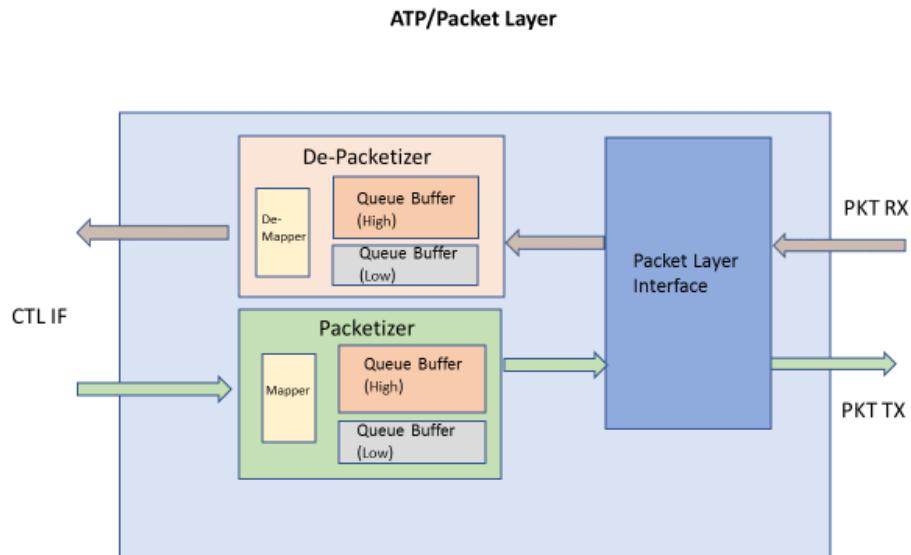
### 6.16.6.3 Arbiter

The arbiter selects one from the available responses to be forwarded to the packet layer based on the arbitration policy selected (parameterized or configured), which are subset of the arbitration algorithms supported for the request arbiter.

- ▶ Round robin
- ▶ Priority with starvation avoidance
- ▶ Deficit Round robin

## 6.17 ATP/Packet Layer (PL)

**Figure 6-18 ATP/Packet Layer.**



The ATP packet layer top-level block diagram is shown in the figure above, and it sends/receives packets to/from the network fabric. For AXI4, if there can be shared/muxed or separate links (request and response) for - read and write (unless its read-only or write-only), and then there is an instance of the PL module per link.

### 6.17.1 Packetizer

The packetizer block interfaces to the CTL interface to receive the messages (req/resp, credit, control etc.) and packetizes them into packet phits (based on the parameters). The parts of the packet are:

- ▶ Link header
- ▶ Network header
- ▶ Transport header
- ▶ Payload (data + meta-data)

**Mapper:** The packet is mapped into PHITs and the header can be one or more PHITs. The first header PHIT contains the link header and optionally the network/transport headers/payload. For multi-PHIT packet, the header PHIT can extend to two PHITS, followed by the payload PHITs. It performs any data width conversion to enable the packing of the data into payload PHITs.

It can optionally do clock domain crossing from ATU to fabric clock before sending the packet out to PL interface module. The idea is to optimally use buffering inside the packetizer to do implementing clock crossing FIFO as well.

Queue Buffers (optional):

There are parameterized queue buffers to hold the packets – High and Low priority, to extend the VC-lite queues into the packet layer of the ATU. Then the packet layer can do the VC-lite handshake with the switch to select packet from one of the queues.

## 6.17.2 De-Packetizer

The de-packetizer block interfaces to the packet layer to receive the messages (req/resp, credit, control etc.) and de-packetizes the packet PHITs (based on the parameters) into messages on the CTL interface.

De-Mapper: It extracts the headers to map the corresponding fields of the CTL message fields, and unpacks the payload to map (and do any data width conversion) to the data bus. It performs any data width conversion to enable the un-packing of the data from the payload PHITs.

It can optionally do clock domain crossing from fabric to ATU clock before sending the message out onto the CTL interface. The idea is to optimally use the buffering inside the de-packetizer for implementing clock crossing FIFO as well.

Queue Buffers (optional):

There are parametrized queue buffers to hold the RX packets – High and Low priority, to extend the VC-lite queues into the packet layer of the ATU. The packet layer can do the VC-lite handshake with the switch to accept an incoming packet into these queues. The de-mapper will arbitrate between these queues to drain the messages to the CT layer.

## 6.17.3 Packet Layer (PL) Interface

The packet interface has a receive and transmit channel. The flow control on the link is done using valid-ready handshake (per VC).

Packet interface signals include the PHIT bus which is parameterized and the side band signals below:

- ▶ Valid (per VC) – PHIT is valid
- ▶ Ready (per VC) – if asserted along with valid, PHIT will be accepted by the next node in the fabric (PIPE/Buffer/Switch/ATU).
- ▶ First – First PHIT indicator
- ▶ Last – Last PHIT indicator
- ▶ ECC/parity – ECC bits or parity bit, calculated per PHIT
- ▶ Panic/Pressure signaling
- ▶ There would some reserved signals as well for future use

## 6.18 Splitting

Request splitting is done at Initiator ATU only, and there is no request splitting at the Target ATU (except for converting into APB request(s)).

The different splitting scenarios are -

1. Splitting based on packet size (max configurable size per ATU)
2. Splitting for address striping
3. Target aware splitting:

- ▶ - address boundary based (like AHB 1KB address boundary)
- ▶ - request/burst size based
- ▶ - protocol specific based (breaking up AXI wrapping bursts)

Request splitting can also be moved into the adapter on the link to be shared amongst multiple Initiators (instead of doing it in ATU).

Any request that gets split, will get the same request ID for the generated split requests but different (incrementing) sequence number to enforce ordering of responses.

The response ordering is handled by reorder buffer (see ROB chapter for details).

## 6.19 Exclusive Transactions:

Exclusive transactions provide a semaphore-type operation without requiring locking of the whole path from the Initiator to the Target.

AXI4, AHB and OCP all support Exclusive Transactions.

To support exclusive transactions Symphony provides monitor logic that can be inserted in the Target ATU.

Exclusive Protocol Definition:

The Exclusive Read/Write protocol is defined as follows:

- ▶ Processor ‘n’ issues an exclusive read (readExcl(‘x’,n)) for memory address ‘x’, where n ={SrcID, ID} and ID = {AXI ID, User Bits} or some combination thereof, and x = {memory address, range}.
- ▶ A corresponding exclusive write (writeExcl(‘t’,n)) to tagged memory address ‘t’ by processor ‘n’ succeeds only if no one else has performed a more recent write to address ‘t’.
- ▶ The slave sends an EXOKAY if writeExcl(‘t’,n) is successful and an OKAY if it fails in AMBA.
- ▶ Exclusive transactions are not cacheable.
- ▶ Note: readExcl(‘x’,n), write(Excl(‘t’,n), EXOKAY and OKAY represent exclusive read, exclusive write, exclusive success, and exclusive fail respectively. Different protocols may have different mnemonics to represent these exclusive commands.

### 6.19.1 Symphony Monitor States:

The Symphony monitor (‘x’,n) has the following two defined states:

- ▶ On: In the “On” state the Exclusive operation is valid.
- ▶ Off: In the “Off” state the Exclusive operation is not valid.

The user has the option to choose whether a T-ATU has the Exclusive transaction monitors or not.

### 6.19.2 Symphony Monitor behavior:

Symphony will support one outstanding Exclusive operation per Master/processor defined as (‘x’, n), where ‘x’ is the memory address and ‘n’ is the Master ID.<sup>1</sup>

Note the description below is mainly based on the AXI4 protocol. Adjustments to support OCP are discussed in a separate sub-section.

Will support transaction with up to 128B.

Exclusive transactions will not be split. The max packet size configured to support the largest Exclusive transaction for the system.

When an Exclusive Read (e.g.: `readExcl('x',n)`), is seen by the monitor, it enters the “On” state and starts monitoring the tagged address ‘x’. The ATU will create an entry in to the table indexed by ‘n’. If multiple processor cores are connected to a single I-ATU, then ‘n’ will reflect the ID of the processor that issues the Exclusive transaction.

When the monitor sees Exclusive write (e.g.: `WriteExcl('t', n)`), it will check the state of the monitor for (`'t',n`). If the monitor is in the “On” state, the Exclusive write will be allowed to update the memory and an ExOKAY message will sent back to ‘n’. Otherwise the Exclusive operation will fail and the following would happen:

- ▶ Memory “t” will not be updated.
- ▶ T-ATU will send OKAY message to ‘n’.

If the monitor sees an Exclusive Write (e.g.: `writeExcl('t',n)`) but cannot find a corresponding entry, the Exclusive write (e.g.: `WriteExcl('t',n)`) would fail and the T-ATU will send an OKAY message to ‘n’ indicating a failure.

If the Monitoring table is full, and a new exclusive read request is received by the T-ATU, the user can configure the following eviction policy at the T-ATU:

- ▶ Do not evict any entry if it is in the “On” state. Exclusive read fails, if no entry can be evicted.
- ▶ Evict the oldest entry that does not have a pending transactions. Fail the exclusive read if no entry can be evicted.
- ▶ Advance eviction policy:
  - ◆ The timer<sup>1</sup> for an entry has expired and there are no outstanding Exclusive Reads or Writes pending for that entry<sup>2</sup>.

If the monitor sees an Exclusive Read, and the T-ATU cannot set up a monitor for that location, then the following steps would happen:

---

## 1. Note: Arm V8 Spec:

A *global monitor* that marks a physical address as exclusive access for a particular PE. This marking is used later to determine whether a Store-Exclusive to that address that has not been failed by the local monitor can occur. Any successful write to the marked block by any other observer in the shareability domain of the memory location is guaranteed to clear the marking. For each PE in the system, the global monitor:

- Can hold at least one marked block.
- Maintains a state machine for each marked block it can hold.

### Note

For each PE, the architecture only requires global monitor support for a single marked address. Any situation that might benefit from the use of multiple marked addresses on a single PE is UNPREDICTABLE or CONSTRAINED UNPREDICTABLE, see [Load-Exclusive and Store-Exclusive instruction usage restrictions](#) on page B2-128.

1. Number of timers is implementation dependent. It is recommended that a single timer be implemented per table. The timer moves to the next eligible entry in Round Robin fashion once it has expired for an entry. Eligible entry is defined as: Entry is in “On” state and has no outstanding Exclusive Reads or Writes pending for that entry.

- ▶ T-ATU will perform the read, but send OKAY response indicating failure of the Exclusive Read.

The monitor can be configured to do the following based on whether the Slave maintains transaction order or not:

- ▶ If the Slave cannot guarantee transaction request/response order between writes to the same address, then T-ATU should be configured to hold any subsequent write/write-Excl('t', n/!n) request till the outstanding writeExcl('t',n) has completed. This is to ensure that outstanding writeExcl('t',n) is not bypassed by a later write to the same address.
- ▶ If the Slave can guarantee write request/response transaction order to the same address, then T-ATU can be configured to not hold subsequent writes to the write-Excl('t',n). In this, the earlier exclusive write will always happen before the preceding write and the responses will be returned in order.

If the monitor('t',n) sees writeExcl('t', !n) and is in “On” state, it will switch to “Off” state.

If the monitor('x',n) sees readExcl('y', n) it will switch to “On” state and start to monitor ‘y’.

Configuration Options:

- ▶ The T-ATU, if configured, will transform a Read Exclusive to a normal read request and Write Exclusive to a normal write. The T-ATU will transform the OKAY response from the Slave for a successful Exclusive Read/Write to ExOKAY response based on the state of the monitor. Else it will send an OKAY message in case of a failure.
- ▶ The T-ATU, if configured, will send the Read/Write Exclusive to the slave if the slave can handle Exclusive transactions. The T-ATU will receive either an OKAY or EXOKAY response from the Slave. T-ATU will then send the appropriate response to the I-ATU based on the response received.

### **6.19.2.1 Monitor State Machine and Monitor Table:**

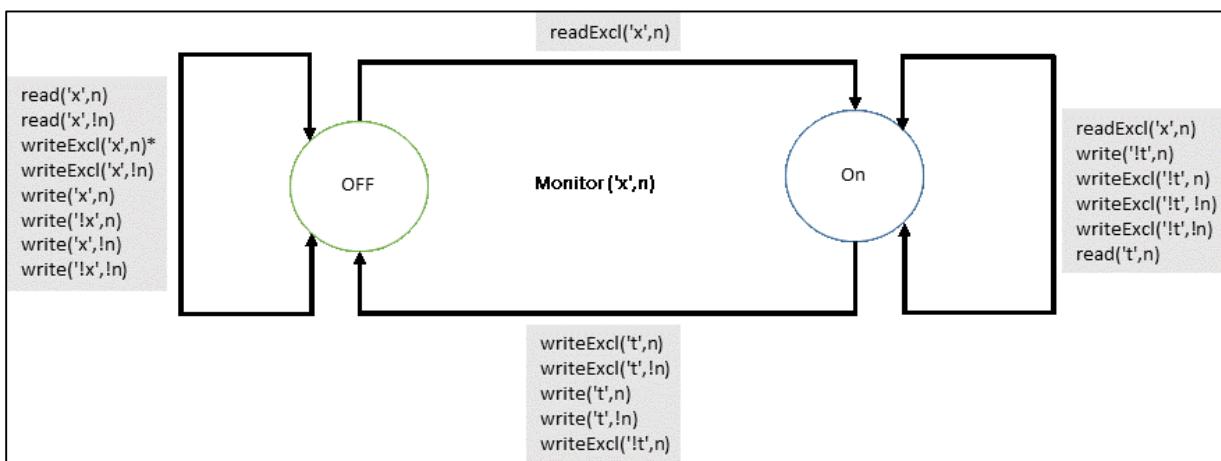
The monitor table is shown below in the Figure.

**Table 6-3**

<b>Source ID</b>	<b>ID</b>	<b>Address</b>	<b>Size</b>	<b>Monitor State</b>	<b>Outstanding Transaction</b>

The The State diagram for the Monitor is shown in Figure below.

- 
2. Note: The timeout is implemented to prevent a situation where the T-ATU is failing exclusive transactions because the Monitor table is full and none of the entries has made any forward progress. It is highly recommended that the Monitoring table be sized to match the number of Masters issuing Exclusive commands.

**Figure 6-19 State Diagram for the Exclusive Monitor ('x';n)**

### 6.19.2.2 AHB and OCP Considerations:

Symphony will generate appropriate commands for the respective protocols. For OCP Symphony will generate a Fail response if the lock fails in response to the WRC (write conditional) command.

For AHB Symphony will generate the HEXOKAY to indicate a success or failure of the Exclusive operation.

In OCP, a WR/WRNP to an address it has previously tagged with the RDL command clear all monitors for other threads except the thread that issued the WR/WRNP. This is slightly different from what the above state diagram depicts. For OCP interfaces, `writeExcl('t',n)` will not change the state from “On” to “Off”.

### 6.19.2.3 Restrictions:

The AXI spec. specify restrictions on the exclusive operation:

- ▶ The addresses for the Exclusive Read and the corresponding Exclusive Write should be identical.
- ▶ The total number of bytes in the Exclusive Read request and the Exclusive Write request should be the same.
- ▶ Number of bytes transferred in an exclusive access burst must be power of 2, that is 1, 2, 4, ..., 128 Bytes.
- ▶ AXI burst length should not exceed 16 transfers.
- ▶ AxCache signal should indicate a non-Cacheable transaction.
- ▶

The Exclusive Read and Exclusive Write should have the same attributes with regards to:

- ▶ Security.
- ▶ Priority.
- ▶ Protection type.

AHB only allows single data transfers.

If these restrictions are violated, an OKAY message will be sent back.

## 6.20 Atomicity:

Atomicity defines the number of bytes the system guarantees to update atomically. Symphony will support Single Copy Atomicity. It will not support Multi-copy Atomicity as Symphony cannot guarantee that all writes to a location are always in order and seen by all Masters. Multi-copy atomicity will be supported by NCORE.

Below are the Atomicity features that Symphony will support and restrictions thereof:

- ▶ Single Copy Atomicity supported only. No Multi-copy support available. AHB supports multicopy. Multicopy atomicity is defined as:
  - ▶ Writes to the same location are observed in the same order by all agents.
  - ▶ A write to a location that is observable by an agent, other than the issuing agent is observable by all agents.
  - ▶ Transactions will not be split below the atomicity, i.e., the maximum packet size should be configured to be  $\geq$  than the support the maximum atomicity size.
  - ▶ Symphony will support different atomicity sizes. It is the responsibility of the system to make sure that all the system components can receive and send transactions can operate at the atomicity size.

## 6.21 Locked Transactions:

AXI4 does not support locked transactions. However, they are supported in AHB, AXI3 and OCP protocols.

Symphony will support locked transactions by locking the path from the I-ATU to the T-ATU.

In a Flow-Through environment the output port has to be locked along the way until the lock is released. In a Flow-Through VC aware network (switches & ATU), the VC and the output port both are locked<sup>1</sup>.

The ATU will indicate a locked transaction to the Fabric. The Fabric will take appropriate actions to lock the path. The ATU indicates the end of the locked transaction by sending a last PHIT with lock signal not asserted. This informs all the network components along the way that the lock should be removed once the Last PHIT of the current packet is serviced.

### 6.21.1 Error conditions:

General error conditions such as address decode error, ECC error etc. are handled elsewhere. This section only deals with errors specific to locked transactions.

---

1. Note: The performance of the system will degrade considerably when locks are provisioned. Since the path from the I-ATU to T-ATU would be locked, any requests destined for other slaves can potentially cause the system to lock up, if back pressure is asserted by the I-ATU on the native interface link.

- ▶ For the AHB interface, if the HAMSTLOCK signal is asserted and a Request is received that is destined for a Target other than the one for which the path is locked, the I-ATU will treat it as an error and handle it in the way it handles other errors. The lock will not be de-asserted.
- ▶ For OCP, once the Thread issues the RDEX command for asserting a lock on the path, any address change before issuing the WR/WRNP to release the lock will be treated as an error by the ATU. The lock will not be released.
- ▶ As soon as the path is locked, I-ATU will start a timer. If the lock is not released before the timer expires, the I-ATU will initiate the process to release the lock. Any subsequent WR/WRNP will be treated as normal write requests and not associated with RDEX command.

## 6.22 Non-Modifiable Transactions:

Symphony will support non-modifiable transaction. The following are the features and restrictions:

- ▶ Symphony will not split Non-modifiable transactions, i.e., the maximum packet size for the Fabric should be configured such to accommodate the largest possible Non-modifiable transaction for the system.
- ▶ The following will not be changed:
  - ♦ Address
  - ♦ Burst attributes as defined by the Specification:
    - ✓ Burst size (unless it is unavoidable)
    - ✓ Burst length (unless unavoidable)
    - ✓ Burst type (INCR, WRAP, etc.)
    - ✓ Others as specified in the AMBA and OCP specifications.
  - ♦ Ordering restrictions and lock type
  - ♦ Any information which if changed would violate the security considerations. These include the AxPROT field in AXI, PPROT in APB, HPROT in AHB and

In certain cases, it would be necessary to modify fields in the transaction, such as a narrower/wider interface on the Slave than the Master, ability to support certain burst sizes, etc. Symphony will change what is necessary to enable the transaction to make forward progress.

## 6.23 Buffered Writes:

Buffered writes transactions can be responded by an intermediate node instead of the final destination, as such the Master IP does not know when the buffered write reached its destination. The AXI specification allows the Master to send buffered writes followed by non-buffered writes using the same AXI ID to ascertain that the buffered writes have reached the destination.

Symphony provides the following options for buffered writes:

- ▶ Option 1:

- ◆ Treat buffered and non-buffered writes the same. That is do nothing. This is lowest cost option, but at the cost of latency for buffered writes.
- ▶ Option 2:
  - ◆ The I-ATU upon receiving the buffered write, will send the write response to the Master, without waiting for the write response to arrive from the final destination<sup>1</sup>.
  - ◆ Master should make sure that buffered writes are visible to all, by sending a non-buffered write with the same AXI ID subsequent to buffered writes.
  - ◆ I-ATU will setup a barrier if a non-buffered write is followed by a buffered write. The buffered write will be allowed to proceed once the response for the previous non-buffered write is received by the I-ATU. That is, if a non-buffered write with the same AXI ID is outstanding, the I-ATU will wait till the non-buffered write response is received before proceeding to send the buffered write.
  - ◆ I-ATU will allow writes with different AXI ID to bypass the barrier. Barrier is setup per AXI ID.
- ▶ Option 3<sup>2</sup>:
  - ◆ The T-ATU responds instead of the I-ATU.
  - ◆ T-ATU will set up a barrier for buffered and non-buffered writes.
  - ◆ If the Slave does not sends an errored response or no response for the buffered write, the T-ATU will raise an error interrupt.

## 6.24 AXI-AHB Bridge Function

### 6.24.11K Burst Crossing:

In AHB the burst should not cross the 1KB boundary. In AXI the bursts boundary is limited to 4KB. In case a transaction originating on AXI side of the interface crosses the 1KB boundary, the I-ATU will split the transaction in to multiple transactions that conform to the 1KB boundary.

### 6.24.2 Protection and Memory Type

The HPORT[7:0] signals in AHB and the AxCache and AxPORT signals in AXI are mapped as follows:

**Table 6-4 Protection Type Mapping**

<b>AXI Signal</b>		<b>AHB Signals</b>		<b>Description</b>
Signal	Value	Signal	Value	

1. Symphony does not guarantee that the buffered write transaction will also get a valid response. In case, an errored response is received for a buffered write, Symphony will raise an error interrupt.
2. This is lower cost than option 2, but may be lower performance due to the possibility of back pressuring the network.

**Table 6-4 Protection Type Mapping**

<b>AXI Signal</b>		<b>AHB Signals</b>		<b>Description</b>
AxPROT[0]	0	H PROT[1]	0	
AxPROT[0]	1	H PROT[1]	1	
AxPROT[1]	0	H NONSEC	0	
AxPROT[1]	1	H NONSEC	1	
AxPROT[2]	0	H PROT[0]	1	
AxPROT[2]	1	H PROT[0]	0	

**Table 6-5 Memory Type Mapping**

<b>AXI Signalling</b>	<b>Value</b>	<b>AHB Signalling</b>	<b>Value</b>	<b>Description</b>
AxCache[0]	0	H PROT[2]	0	Non-Bufferable
AxCache[0]	1	H PROT[2]	1	Bufferable
AxCache[1]	0	H PROT[3]	0	Non-Modifiable
AxCache[1]	1	H PROT[3]	1	Modifiable
AxCache[2:3]	00	H PROT[5]	0	No-Read/Write Allocate
AxCache[2:3]	11	H PROT[5]	1	Allocate

Shareable and look up bits are not carried across. AHB coherency domain does not cross in to the AXI domain.

### 6.24.3 Address Mapping

AHB supports 32 bit address. AXI4 supports 64 bit address. Therefore the memory maps between AHB and AXI may vary. Symphony allows hierarchical address mapping where the next stage of address decode can be done at the T-ATU. Transactions entering the system via the AXI interface will perform the address look up to determine the Target ID of T-ATU. Complete or compressed address will be shipped along the packet to the T-ATU. The T-ATU will then look up the address to determine the AHB Slave on the AHB bus.

If it is a single AHS slave hanging off the T-ATU, then no-address decode would be required in the T-ATU.

On the reverse direction (AHB->AXI4), the 32 bit address would be mapped to an equivalent 64 bit address, compressed or otherwise by the I-ATU. Hierarchical address decode could be done, if needed.

### 6.24.4 Data Width Adaption

Treated as any other data width adaption.

## 6.24.5Timeout

T-ATU and I-ATU have multiple timeout counters to track outstanding transactions. Details of how the timeout is handled is TBD.

## 6.24.6Burst Mapping

Burst mapping from AXI4 to AHB.

- ▶ Incrementing burst of length 1 is converted into the equivalent AHB single transaction.
- ▶ Incrementing burst of length 4, 8, and 16 are converted into equivalent transactions on the AHB interface as long as it does not cross 1KB boundary.
- ▶ Any incrementing burst of length up to 256 other than 4, 8 and 16 is converted in to AHB incrementing burst of infinite length.
- ▶ Wrapping burst of 4, 8, and 16 are converted in to equivalent AHB wrapping bursts.
- ▶ Two AHB Transactions are generated whenever an AXI transaction crosses 1KB boundary.
- ▶ Wrap burst of 2, is converted to two AHB single burst transactions.
- ▶ There are no Fixed burst transactions in AHB. Fixed burst transaction is converted to single transactions to the same address.

Burst mapping from AHB to AXI4:

- ▶ A single burst transfer is converted to AXI burst transfer for 0 increment.
- ▶ Incrementing burst of infinite length is converted to AXI semantic.
- ▶ WRAP4/8/16 on AHB is converted to an equivalent WRAP on the AXI side.
- ▶ INCR4/8/16 on AHB is converted to an equivalent INCR on the AXI side.

## 6.25 AXI-APB Bridge Function

### 6.25.1Burst Crossing

Same as in See “Burst Crossing” on page 126.

### 6.25.2Protection

APB provides PPROT[2:0] to identify transactions that are secure/non-secure, normal or privileged, and data or inst. These signals are mapped to AXI signals as follows:

<i>AXI Signal</i>		<i>APB Signals</i>		<i>Description</i>
Signal	Value	Signal	Value	

<b>AXI Signal</b>		<b>APB Signals</b>		<b>Description</b>
AxPROT[0]	0	PPROT[0]	0	Unprivileged
AxPROT[0]	1	PPROT[0]	1	Privileged
AxPROT[1]	0	PPROT[1]	0	Secure
AxPROT[1]	1	PPROT[1]	1	Non-Secure
AxPROT[2]	0	PPROT[2]	0	Data
AxPROT[2]	1	PPROT[2]	1	Inst

**Table 6-6 AXI-APB signal mapping**

## 6.25.3Address Mapping

Same as See “Address Mapping” on page 127.

## 6.25.4Burst Mapping

APB supports single transfers. Hence all bursts from AXI will be mapped to single transfers on APB. Transactions will be split in the I-ATU as well as in the T-ATU depending on the size, and the amount of buffering available at the T-ATU.

## 6.26 OCP Handling

Full discussion on OCP native layer will be added post 0.7 release. However, we will discuss how Symphony will handle OCP transactions at a high level and in particular discuss the transaction ordering and thread flow control management.

### 6.26.1OCP Ordering

OCP ordering rules as stated in the OCP Specification Release 3.0 are as follows:

- ▶ All writes, without Tags, from a Master should be committed in-order.
- ▶ Without Tags, a Slave must return all responses in the order the corresponding requests were issued by the Master.
- ▶ With Tags, the responses can be returned out of order. However, order has to be maintained between response with the same TagID.
- ▶ Write with different TagIDs can be committed out of order.

If Tags are not used in OCP, then the requirement is that all transactions issued by the Master should be ordered, both in terms of being committed to the Slave and also in terms of the response coming back.

Symphony will inherently preserve order between writes if they are mapped to the same Flow (See Flow definition).

Symphony inherently does not maintain order between Read and Write transactions, in that sense it follows the AXI ordering model. However, Symphony will maintain order if both read and write transactions are mapped to the same Virtual Path (single request and data network). Symphony will add sequence numbers so that responses can be reordered at the I-ATU.

Alternatively, if the reads and writes use separate interfaces, then ordering between them is not guaranteed. In this case, Symphony will establish a barrier so that the preceding write/read completes (response is received by the I-ATU) before the subsequent write or read is sent.

Depending on how Tags are used by the System, Symphony may not need to put up a barrier to maintain order between Reads and Writes.

## **6.26.2 Thread Management:**

According to OCP Spec 3.0, there are no ordering requirements between transactions belonging to different threads. Threads have dedicated flow control, that is, each thread can independently flow control the transactions that are mapped to that Thread. Transactions within the Thread have to remain in order unless they are tagged.

The Spec also states that Threads terminate at the Bridging device, that is, the Thread semantics need not be carried all the way to the Slave.

Symphony will enable 1:1 or N:1 mapping between Threads and Virtual Channels in the Interconnect. However, Symphony Native Layer will assert back pressure and respond to back pressure from the Master on a per Thread basis.

## **6.27 ARM Kite<sup>1</sup> Processor Interface:**

The Kite Processor has the following external interfaces:

- ▶ AXI Master, AXI Slave, Low Latency Peripheral Port (LLPP), and Flash

All ports are based on the AMBA AXI4 protocol.

Kite has two optional bus protection schemes:

- ▶ Signal Integrity Protection
- ▶ Interconnect Protection
  - ◆ Universal Transaction ID
  - ◆ Even/Odd Beat Indicator (EOBI)

### **6.27.1 Signal Integrity Protection**

Signal integrity protection enables detection and correction, in some case, of errors that occur on the signals transmitted on the above buses. In general, the following protections are provided:

- 
1. The Kite processor uses the ARMv8-A architecture. It enables a bare metal hypervisor mode allowing real time and non-real time OS to operate on the core at the same time. The reason why the processor is called out specifically in the Arch doc is because of the optional ECC, parity and UTID (Universal Transaction ID) features it introduces for reliable Automotive application.

- ▶ Single-error and double error detection on control, address and response payload signals
- ▶ Single-error correction and double error detection on data payload signals.
- ▶ Parity protection for single-error detection on handshake signals.

The table below lists signal integrity protection for various signals:

**Table 6-7 AxRequest signals**

<b>AXI4 Signal</b>	<b>Width</b>	<b>Bus Protection Signal</b>	<b>Protection Scheme</b>	<b>Symphony</b>
AxID	ID_Width	AxIDCode[4:0]	Single and double detection only	Single bit correction/ Double bit detection
AxPROT	3	AxCTL0Code[4:0]		
AxCACHE	4			
AxQOS	4			
AxLEN	8	AxCTL1Code[4:0]		
AxSIZE	3			
AxBURST	2			
AxLOCK	1			
AxADDR	ADDR_WIDTH	ARADDRCODE[6:0]	Single bit correction double bit detection only	Single bit correction/ Double bit detection
AxVALID	1	ARVALIDCODE	Odd parity only	Parity/ECC/TMR
AxREADY	1	ARREADYCODE		

**Table 6-8 Read response/Write Response/Write data signals**

<b>AXI4 Signal</b>	<b>Width</b>	<b>Bus Protection Signal</b>	<b>Protection Scheme</b>	<b>Symphony</b>
RID/BID/WID	ID_Width	RIDCODE[4:0]/BID-CODE[4:0]/WID-CODE[4:0]	Detection only	SECDED
RRESP/BRESP	2	RCTLCODE[2:0]/BCT-LCODE[2:0]		
RLAST/WLAST	1	RCTLCODDE[2:0]		
WSTRB	STRB_WIDTH	WCTLCODE[2:0]		
RVALID/BVALID/WVALID	1	RVALIDCODE/BVALID-CODE/WVALIDCODE	Odd parity only	Parity/ECC/TMR
RREADY/BREADY/WREADY	1	RREADYCODE/BREADYCODE/WREADYCODE		

**Table 6-8 Read response/Write Response/Write data signals**

<b>AXI4 Signal</b>	<b>Width</b>	<b>Bus Protection Signal</b>	<b>Protection Scheme</b>	<b>Symphony</b>
RDATA/WDATA	DATA_WIDTH	RDATACODE[x-1:0]/WDATACODE[x-1:0]	Detection and correction	SECDED

The I-ATU will terminate the interface signals and add Symphony based protection inside the Fabric. The T-ATU will convert Symphony based protection back to ARM v.8 specific protection signals. On the return path same will be done only in the reverse direction.

## 6.27.2 Interconnect Protection

There are two schemes that are optionally recommended by the ARMv.8 specification “Kite Processor Integration Manual for Interconnect Partners, Rev r0p0”, these are:

- ▶ Universal Transaction ID (UTID)
- ▶ EOBI signaling. The limitation of EOBI is that it assumes that the transaction propagate through the Interconnect without being modified.

UTID is constructed as follows:

**Table 6-9 UTID mapping**

<b>Range</b>	<b>Description</b>	<b>Symphony Mapping</b>
UTID [9:8[	Cluster ID	SrcID
UTID [7:6]	Core ID	MsgID
UTID [5:4]	Port ID	
UTID [3:0]	Transaction ID	

## 6.27.3 Virtual Machine ID (VMID)

ARM v.8 specification (Sept 25, 2017) supports the Hypervisor running in the privileged EL2 mode. Virtual machines run in the EL1 and EL0 mode. The Hypervisor assigns a virtual machine identifier (VMID) to each virtual machine.

The ARM Cortex R52 processor guarantees a valid VMID for device memory access only. It defines two signals: ARVMIDMx and AWVMIDMx that read and write the VMID. These signals come out on User signals.

Symphony, will map these VMID signals in to the VNID values. The VNID values are configured by the system based on the overall number of VMs in the system.

## 6.28 AXI ID, QoS and Transaction Ordering:

## 6.29 AMBA AXI 5<sup>1</sup> Support:

AXI5 extends the capabilities of the AXI4 protocol to include the following:

- ▶ Atomic Transactions.
- ▶ Data Check.
- ▶ Poison.
- ▶ QoS accept.
- ▶ Trace signals
- ▶ User Loopback.
- ▶ Wakeup Signals.
- ▶ Untranslated transactions.
- ▶ Non-Secure Access Identifiers.

This section will detail how each of these are handled in Symphony and the ATU in particular.

### 6.29.1 Atomic Transactions

The following Atomic Transactions are supported by Symphony. Symphony supports in providing the appropriate transport for these transaction.

- ▶ AtomicStore.
- ▶ AtomicLoad.
- ▶ AtomicSwap.
- ▶ AtomicCompare.

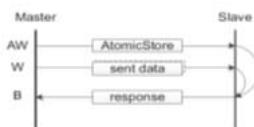
Atomic Transactions have the following restrictions:

2. AXI IDs for Atomics cannot be shared by regular transactions. That is, if there is either an Atomic or a regular transaction outstanding then that ID cannot be shared with the other “type” of transaction. This ensures there are no ordering constraints between Atomic and non-Atomic transactions.
3. A single AXI ID is used for an Atomic Transactions: request, write response and read data.
4. Atomic transaction cannot be broken up in to multiple packets.
5. Error response is generated if the address or the device that does not support Atomics receives an Atomic transaction.
6. Multiple Atomic transactions that are outstanding must not use the same AXI ID value.

#### 6.29.1.1 AtomicStore:

The Master sends a single data value with an address and the atomic operation to be performed to the Slave. The Slave performs the operation using the data sent and the value at the addressed location. The

result is stored at the addressed location and a single response without data is sent back to the Master. The outbound data size can only be 1, 2, 4, or 8 bytes.



**Figure 6-20 Atomic Store (Ref: Fig E2-2, AMBA5 Protocol Spec. IHI0022F.b)**

Symphony support options will depend on the following conditions/assumptions:

1. I-ATU:
  - a. Master guarantees that Atomic transactions will never use the AXI IDs in use by Normal Transactions.
  - b. The Master does not guarantee AXI ID separation and the I-ATU has to setup a barrier.
2. T-ATU:
  - a. No ID remapping or compression is required at the T-ATU (rare case)
  - b. ID remapping/compression required at the T-ATU.

### 6.29.1.1I-ATU:

If the Master guarantees that Atomic and non-Atomic transaction IDs never clash, the I-ATU does not need to set up a barrier other than to make sure that there is only one AXI ID outstanding for Atomic Operation. The context table will include the following information:

1. Transaction AXI ID
2. Transaction attribute definition at the CTL layer provides information about type of transaction (Store, Load, Swap, Compare, Normal Read, Write, Multicast). Note: Transaction type information is available to the CTL layer.
3. Number of transactions outstanding for each AXI ID. Only one outstanding Atomic transaction is allowed per AXI ID. This will set up the barrier.

In the above case, the I-ATU can setup either a dedicated context or shared context for Atomic Transactions [implementation dependent]. In either case the I-ATU will check if there is an outstanding transaction pending for the AXI ID. If it is, the I-ATU will hold the Atomic transaction till the outstanding transaction is complete.

Whether the Target device or Target address supports Atomic Transactions would be part of the PAM. If Atomic transactions are not supported by the device or the address, an error response will be generated by the I-ATU and send back to the Master.

If the Master does not guarantee that there would be no clash between the AXI ID for Normal transactions and Atomic transactions, the I-ATU will setup a barrier by searching the write context and in some cases read context as well for a conflict. If a conflict is detected in either Write or Read context, the transaction will be held until the outstanding transaction is complete. Note: for Atomicload and AtomicSwap the read response should have the same AXI ID as the write request. Holding the transaction will back pressure write requests and for AtomicLoad, AtomicSwap and AtomicCompare the read requests as well.

As the response comes back, it is looked up. Depending on the type of Atomic transaction type, the response can be accompanied with data as well.

## 6.29.1.1.2T-ATU:

If no ID remapping or compression is needed at the T-ATU, the T-ATU treats the transactions like any other Normal transaction, except for AtomicSwap, AtomicCompare and AtomicLoad. The T-ATU will have a separate Atomic context as these transactions are special.

If ID remapping is required, then the T-ATU can be configured to have one of the two configurations:

1. Shared IDs:

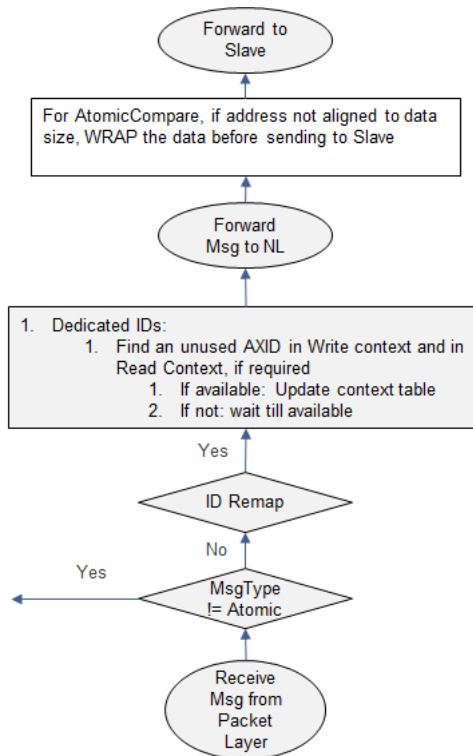
- a. The IDs are shared between Normal and Atomic Transactions. This option will necessitate that a barrier be setup in case matching free IDs in Write and Read context are not available.

2. Dedicated IDs:

- a. This is the preferred option as it will result in better performance. A separate context for Atomic write and read transactions will track ID usage and outstanding transactions. Only one outstanding transaction is permitted per ID.

The flow chart for packets received at the T-ATU from the network is shown in Figure below:

**Figure 6-21 Flow Chart for when Packets are received from the network at T-ATU**



## 6.29.1.1.3Protocol Field Mapping

The AWTOP[5:0] field in the AXI5 protocol indicates whether the transaction is an Atomic transaction or a normal. The AWTOP field maps in to the MsgType field in the packet header (see Packet Protocol Chapter, Section 5.3.2).

The AWTOP is also carried in the TxHdr field of the packet for the Slave to be able to perform the appropriate operation.

Certain fields can be optimized in the TxHdr field as they can be inferred from the MsgType field in the packet header.

For AtomicCompare transactions, the T-ATU will adjust the order of data according to whether the address is aligned to the total size of the outgoing data.

The T-ATU will accordingly adjust the AWLEN if AWLEN field was greater than 1 in the Compare transaction.

For AtomicStore, load and Swap transactions, the burst type is INCR. For AtomicCompare, if the address is aligned to the transaction size, the burst size is INCR otherwise it is WRAP.

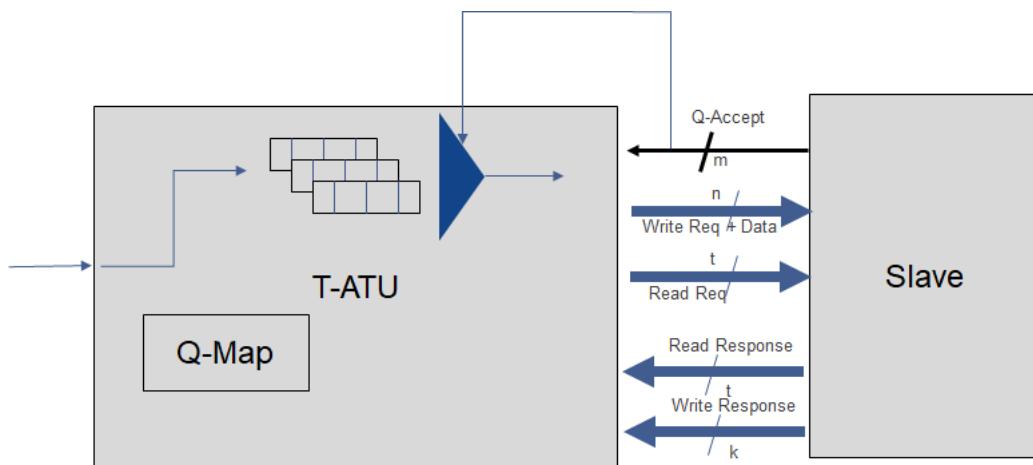
## 6.29.2AXI5 QoS Accept Signals

AXI5 adds two more signal interface to AXI4 QoS signals, to indicate the lowest QoS value it cannot accept transaction for. This signal is useful for memory controllers to indicate to the master if lower QoS buffering is congested and not able to accept any more transactions. The signals are dedicated for read and write transactions.

In Symphony, the T-ATU will terminate these signals. The T-ATU will forward transactions to the Slave based on the value of the signals. T-ATU will map packets based on the QoS value to the appropriate queues in the CTL layer. The CTL layer will also map these QoS queues to appropriate QoS values on the Slave and forward packets/Transactions based on the instructions received from the Slave via the QoS signals.

The Figure below shows how the QoS Accept signals will be used by the Queue Arbiter in the T-ATU to arbitrate.

**Figure 6-22**



## 6.30 Error Handling:

1. Request timeouts (both Initiator and Target ATU).
2. Address Map decode error.
3. Protocol checking errors.
4. Resiliency checking errors.
5. Buffered write response error.

1. Based on “AMBA AXI and ACE Protocol Specification, version IHI 0022F.b. Release Date: 21 Dec, 2017.



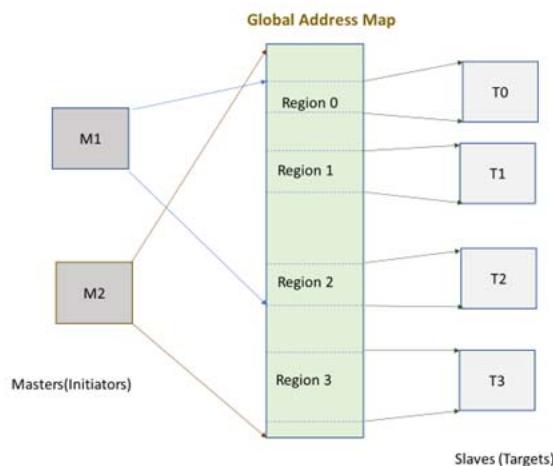
## 7

# Memory Map

This chapter describes the Global Address Map (user input to software) and Partial Address Map (software generated address map for each initiator).

## 7.1 Global Address Map

User specifies the Global Address Map (GAM) and describes how every Masters and Slaves Map into it. This specification is handled and interpreted by Software. The GAM is described here for completeness and to describe any limitations imposed by the Interconnect components.



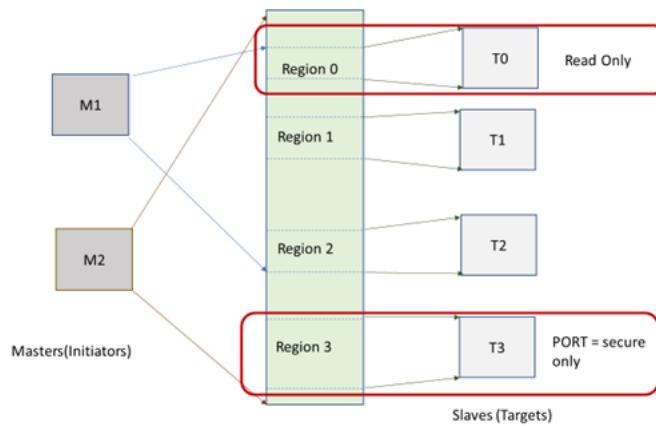
**Figure 7-1 Global Address Map Example.**

**Global addresses** – The concept of global address is for NoC level, not at Master or Slave level. The set of global addresses encompasses the set of all addresses that all the masters can generate.

**Region** – Set of global contiguous addresses that map uniquely to one slave (or an aggregate of multiple identical slaves – for address striping).

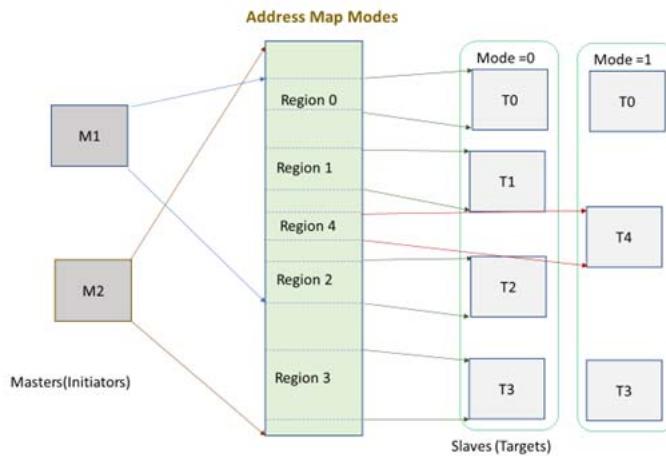
- ▶ A Master's (Initiator) addressable space can be composed of one or more address regions.
- ▶ Each region size is power of 2, starting address is aligned to its size.
- ▶ Multiple regions can map to same Slave.
- ▶ There can also be holes (no region) in the Global/Master's address space.

- ▶ Each region's mapping to its target address space can have a configured mapping – that is its base address is configurable by user. Different region's (from different Master's) can be mapped to same target address space.
- ▶ The regions can overlap and in the overlap address space – the selected region is the one that is first in the partial address map (also, a single request is not permitted to cross regions).
- ▶ Each region can have optional attributes –
  - ◆ Request type - Read only, Write only, Read/Write (default)
  - ◆ In band qualifier – user bits, protection etc.
  - ◆ Virtual network ID



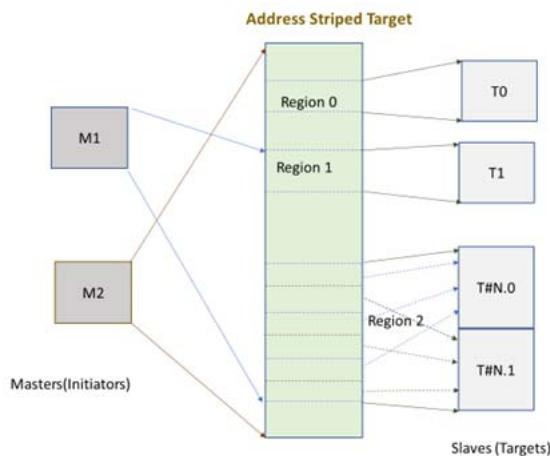
**Figure 7-2 Region Attribute Example**

- ▶ There can be more than one address map, one per each mode (based on mode select inputs into NoC) like -
  - ◆ Regular/default mode
  - ◆ Boot-up mode
  - ◆ Secure mode
  - ◆ User mode



**Figure 7-3 Address Map Mode Example**

- ▶ A slave can be an aggregate of multiple identical slaves (address striping)<sup>1</sup>
  - ◆ Each of these identical slaves is a separate target
  - ◆ Target selection is done using  $\text{Log}_2(\text{number of targets})$  address bit selected by the user. The other option for target selection is to have custom striping logic.
  - ◆ Support – 2/4/8/16 targets
  - ◆ Stripe size as low as 1 slave word (or master word, whichever is largest)



**Figure 7-4 Address Stripping Example**

#### 1.R1 release limitations -

Minimum region size is 4KB

For address striping – support only power of 2 targets, and  $\text{Log}_2(\text{targets})$

) address bit based target selection (unless there is other requirement from the lead customer)

## 7.2 Partial Address Map

The Partial address map (PAM) is the address map used by initiator ATU (or adapter) and is generated by the SW based on the Global address map. The generated values (hard-wired defines/parameters) are fixed, not configurable. The address bits and ranges in PAM can be subset of those in the GAM.

Conversion (by SW) of a region in GAM to PAM consist of –

Only regions corresponding to a given Master's address space are populated in PAM

Convert global address to local address for that Master

Each entry in PAM table consist of –

- ▶ Base address and size: The size must be power of 2, and the base address should be aligned to the size of the address range. The minimum size for address range for a target is 4KB (R1 limitation).
- ▶ Target ID: The assigned target\_ID (by SW) for the address range
- ▶ Mapped Target Base: The base address mapped into target's address space
- ▶ Striped: Index to stripe group table if the address range is striped. The stripe group table consists of details to handle the address striping – the number of striped targets, the respective target\_IDs and routes, striping function select and stripe size.
- ▶ Route: Routing bits for the route to get to the target (determined by SW)

Like GAM, there will be separate PAM per addressing mode (based on mode select inputs into NoC) –

- ▶ Regular/default mode
- ▶ Boot-up mode
- ▶ Secure mode
- ▶ User mode

The figure below shows an example PAM with parameters - (PAM\_ADDR\_WIDTH =32, PAM\_ENTRY\_CNT=5, PAM\_MAP\_BASE\_EN=1, PAM\_RW\_CHK=1, PAM\_STRIPE\_EN=1, PAM\_STRIPE\_GRP\_CNT=2, ROUTE\_BIT\_WIDTH = 12)

**Table 7-1 Partial Address Map Table**

<b>Base Address</b>	<b>Size (Power 2)</b>	<b>Target ID</b>	<b>Mapped Base Address (Optional)</b>	<b>Route</b>	<b>RW (Optional)</b>	<b>Stripped (Optional)</b>	<b>Stripe Group (Optional)</b>
0x0000_8000	0x4000	1	0	12'h011	RW	0	X
0x0001_0000	0x1_0000	4	0x10000	12'h020	R	0	x
0x4000_0000	0x4000_0000	X	X	X	RW	1	0
0x8000_0000	0x4000_0000	X	X	X	RW	1	1

**Table 7-2 Stripe Group Table**

<b>Stripe Group</b>	<b>Stripe Target Count</b>	<b>Target ID</b>	<b>Mapped Base Address (Optional)</b>	<b>Route</b>	<b>Stripe Func_sel</b>	<b>Stripe Size (power of 2)</b>
0	2	{10,14}	{0,0}	{12'h040,12'041}	0	0x400
1	4	{11,12,13,15}	{0,0,0,0}	{12'h042,12'h043,12'h044,12'h045}	1	0x100

Address = offset, where offset = (start address – Base address), OR

Mapped base address + offset (if Mapped base address exists), OR

Mapped stripe address (if address striping)

#### **Striping:**

An address range in the PAM table that is marked as striped, can stripe the address across multiple targets, and the striping group provides the information to handle the striping-

- ▶ Stripe Count – number of targets striped (NUM\_STRIPE\_TARG)
- ▶ Stripe size – power of 2 (STRIPE\_SIZE),
- ▶ size as low as 1 slave word (or master word, whichever is largest)
- ▶ Stripe function (STRIPE\_FUNC\_SEL) -
  - ◆ select the starting bit number to index the target selection bits (power of 2 – log(NUM\_STRIPE\_TARG))
  - ◆ Customized striping: A drop-in customized/library module that determines the target (not limited to power of 2 targets)

A request spanning across striping boundary, will be split and the responses have to be re-ordered or interlocked.

#### **Mapping:**

The striped addresses to a target are not contiguous and can use address mapping to make it look contiguous/ fill up the address gaps. The address mapping feature is optional, and can follow the mapping options (parametrized, STRIPE\_MAP\_EN) –

- ▶ Mapping for striped bits (STRIPE\_MAP\_FUNC) –
  - ◆ Shift right the bits above the striping bits by number of striping bits
  - ◆ Move the MSB bits to replace the bits used for striping.
- ▶ Customized Mapping- A drop-in customized module that determines the address given the target selection (It can be part of the customized striping module or separate).

#### **Route information:**

The routing information for a given target is fixed (can be considered to be made configurable post R1) and stored as part of the PAM table. The route lookup gets done as part of the target ID lookup.

**Target Aware Request Split:**

There can be optional columns in the PAM for target aware request splitting. There are parameters to enable request splitting per target for –

- ▶ Boundary based (like AHB – 1KB boundary)
- ▶ Size based (limit to max burst that target can accept)
- ▶ Protocol capability based – split if target does not support wrap or has smaller burst support

The fields (optional) that can be added per target entry in PAM are –

- ▶ Split\_boundary: the address boundary to split request (PAM\_TARG\_SPLIT\_BDRY), enabled by PAM\_TARG\_SPLIT\_BDRY\_EN
- ▶ Split\_size: the burst size to split request (PAM\_TARG\_SPLIT\_SZ ), enabled by PAM\_TARG\_SPLIT\_SZ\_EN
- ▶ Split\_wrap: split wrap (PAM\_TARG\_SPLIT\_WRAP), enabled by PAM\_TARG\_SPLIT\_WRAP\_EN

**Table 7-3 PAM Splitting**

Target_ID	PAM_TARG_SPLIT_BD RY	PAM_TARG_SPLIT_SZ	PAM_TARG_SPLIT_W RAP
1	0	0	0
4	1024	128	1
2	0	256	0

## 7.3 Error Conditions:

TBD

# 8

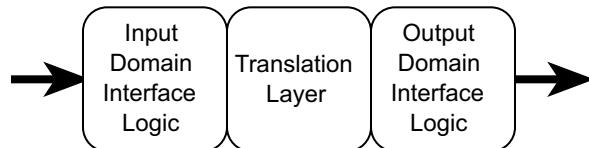
# Adapters

Adapters bring messages up from the transport layer to the network layer, perform a translation and return the message(s) to the transport layer. For a detail explanation for the Packet Protocol Layers refer to Chapter Arteris Transport Packet Protocol on page 35

## 8.1 Adapter

An adapter is a device that sits on a link and adapts communication attributes from one side to other. An adapter is inserted into a link thereby splitting the link into two parts. A high level block diagram is shown in Figure 8-1 on page 143

*Figure 8-1 Adapter Block Diagram.*



### 8.1.1 Adapter Functions

Adapters can perform the following:

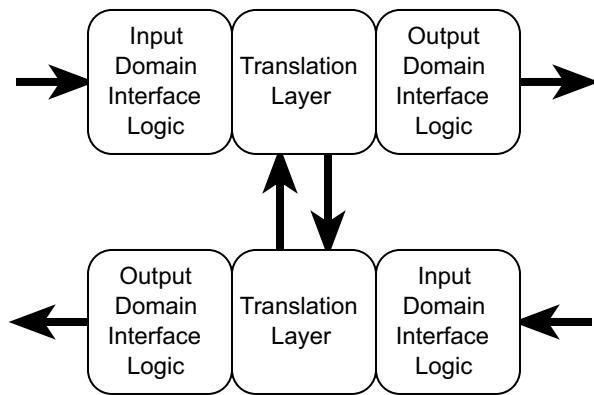
- ▶ Width changes
- ▶ Address to target decode
- ▶ Clock changes
- ▶ Voltage changes
- ▶ Connections between fabrics
- ▶ VC remapping logic

## 8.2 Bidirectional Adapter

A bidirectional adapter is a device that sits on two links that have the property that all responses to requests on one link returns on the other link. This does not imply that one link is dedicated to request and the other to responses. A bidirectional adapter conceptually is composed of two adapters whose

translation layers communicate with one another. A high level block diagram is shown in Figure 8-2 on page 144

**Figure 8-2 Bidirectional Block Diagram**



## 8.2.1 Bidirectional Adapter Functions

Bidirectional Adapters perform the following functions:

- ▶ Connections to clock domains
- ▶ Connections to power domains
- ▶ Contain reorder buffers
- ▶ Error reporting and response

---

*Conceptually a Bidirectional Adapter can perform any function that can be performed by a target ATU connected to an initiator ATU of the same native protocol. In reality, it can perform more, because it's not restricted by the conventions of the native protocol.*

---

## 8.2.2 Between Fabrics

In a Symphony deployment, different portions, or domains, of the interconnection solution might need to be tuned to different points of the PPA space, saving area and power where possible, increasing performance at added cost where necessary. Different styles of switches would be used in these different domains.

Yet, it must be possible for two end-point devices to communicate regardless of where they are situated in the overall interconnect and whether multiple distinct domains of switch styles lie between them. The messages flowing between them must be able to traverse seamlessly across multiple domain styles.

The transmission between domain styles is made possible by adapters that are located between the domains

These types of adapters are limited to cases wherein

- ▶ Both Input and Output domains are Symphony domains, and
- ▶ No modifications are made in the adapter that affects the contents of the data-link layer header of the packet.

### **8.2.2.1 Configuration Adapter**

The configuration Adapter is a bidirectional adapter that will be attached to a regular Fabric on the upstream side and to the configuration Fabric in the downstream side.

### **8.2.3 Error Reporting**

TBD.



# ReOrder Buffer

**The ReOrder Buffer (ROB) function is to reorder the responses as required per the ordering rules. This component will be implemented as function of bidirectional adapter. The ROB can be inserted on any link with the requirement that both the request and response link go through it.**

The functional modes that the ROB can support are -

1. Reorder all responses (parameterized - read, write, read and write) based on the order the requests are received. This also acts as a de-interleaver to support de-interleaving of responses if the external master does not support AXI read response interleaving.
2. Reorder the responses (parameterized - read only, write only, read and write) based on AXI reordering requirements for one Initiator/Master -
  - a) Requests with same AXI ID, need to have their responses return in order.
  - b) An input request that gets split/stripped into multiple requests inside the NoC, the corresponding responses need to be in order.
  - c) Requests with different AXI ID can return responses out of order.
3. Same as #2, but support reordering for multiple initiators/masters. This sharing of the ROB across multiple masters can greatly reduce the buffer cost per master.

---

**Note**

The Reorder buffer can also be used for data reassembly. The cost will be high as the reorder buffer will also include a data width adapter.

---

## Details -

ROB consist of pending request context and buffer to hold out of order responses. Based on the NoC configuration, software will determine where to insert the ROB (exclusive or shared between masters) and the depth of the buffer (trade-off pending requests vs. buffer cost).

## For AXI ordering -

Initiator ATU adds a tag - sequence number for the requests, and the ordering is based on {req\_id, seq\_num} per requester. There is a request context that stores the ordering ID per pending request, and if there is a ordering relation for a new request it gets allocated space in the buffer (to hold response if it comes back out of order). If there are multiple sources, the ordering is based on {src\_id,req\_id,seq\_num}.

The requests that belong to different sources have no ordering relation to each other.

The requests with different req\_ID have no ordering relation to each other.

The requests with same requests req\_ID and src\_ID, need to be have responses ordered in order of the sequence number. The requests with buffer space allocated, get the responses stored in the buffer if the ordering requires the response to be held (other related response still pending). The response from the buffer can be read out once the ordering dependency is removed.

The implementation details will be covered in the micro architecture spec for the ROB. The buffer can be register based or memory based (likely to be memory). For memory based buffer, there will be additional latency to read and write from/to memory and may need additional pipeline for timing requirements.

# 10

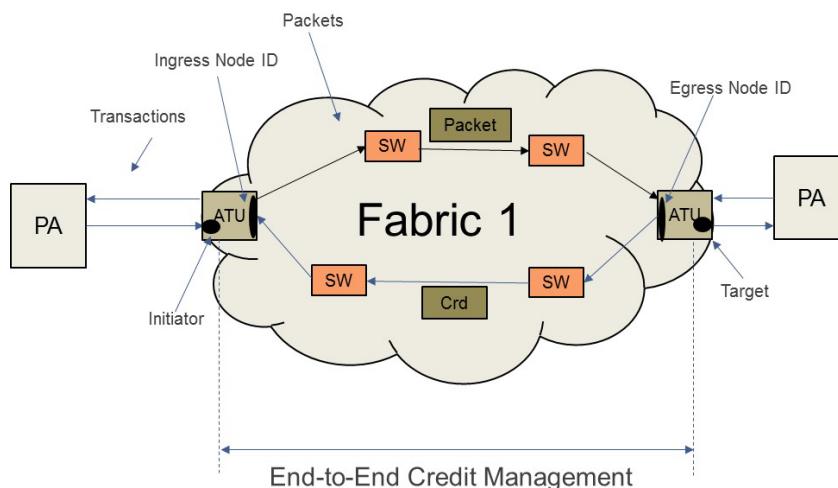
## End-to-End Credit Management

The End-to-End Credit Management scheme, also referred as Arteris Credit Management (ACM), is designed to achieve the following goals:

- ▶ Regulate the traffic injection rate at the Initiator ATU (I-ATU) by providing feedback on the congestion in the network.
- ▶ Ensure high bandwidth utilization of the link connecting the Target ATU (T-ATU) and the Slave.
- ▶ Enable the system Architect to pre-allocate credits to requesting Initiators based on the QoS policies and the bandwidth requirements of the I-ATU.
- ▶ Enable the T-ATU to regulate credit return to the I-ATUs based on congestion and service policy at the Slave.
- ▶ Minimize credit request/response traffic between I-ATU and the T-ATU.
- ▶ Provide a robust protocol.
- ▶ Reduce the overall credit buffering requirement at the Target ATU.
- ▶ Interface with memory schedulers that support QoS policies to provide end-to-end QoS in the system.

In ACM, credits are initiated and terminated at the Initiator ATU and the Target ATU respectively. The switches and other elements in the Fabric do not interpret the credit control messages other than to route them or pass them on to the appropriate destination.

**Figure 10-1 Symphony End-to-End Credit Management Architecture**



## 10.1 Features

Key features of the ACMs are as follows:

- ▶ Symphony will provide the following configuration options for sizing the buffers in the T-ATU:
  - ◆ No link back pressure. In this case buffering is required for all outstanding transactions.
  - ◆ Buffering based on effective bandwidth of the slave.
  - ◆ Limited back pressure. In this case the buffering required is determined by the acceptable probability of back pressuring the link and the effective rate of the slave. The ACM decouples credit from the buffering required at the T-ATU in this case. As a result, the buffers required at the T-ATU are calculated based on amount of outstanding credits, the effective bandwidth of the slave, and the acceptable back pressure probability.
- ▶ Credit and non-Credit domains can co-exist in the network and if required within the same I/T-ATU.
- ▶ Flexibility in assigning credits and managing states:
  - ◆ The scheme allows for credits to be dedicated and/or shared between ATUs.
  - ◆ The scheme allows for pre-allocation of credits to Initiators that require low latency. I-ATUs that are pre-allocated tokens do not have to request tokens and wait the round trip time (RTT) before they can send packets in to the Fabric.
  - ◆ The scheme also allows for credits to be given to the Initiator on-demand, that is, for on-demand initiators, credits are not pre-allocated but are given when requested by the Initiators.
  - ◆ Additional credits can be requested by the Initiator ATU dynamically if the credits initially assigned were either not sufficient.
- ▶ The scheme is designed to reduce extra traffic generated due to the credit exchange protocol. Two types of credit messages are supported:
  - ◆ Credit requests/grants embedded in regular request/response transactions.
  - ◆ Independent standalone credit messages.
  - ◆ A single credit message whether embedded or independent can carry request/grant for multiple credits.

## 10.2 Credit Management Scheme

The Credit Management scheme as stated above, is an end-to-end mechanism and does not require the network elements in the Fabric to interpret the credit messages.

Credits are assigned to the credit tuple as defined by the set: {I-ATU, T-ATU, VNID, QoS}. Implementation may choose to ignore the VNID and QoS fields, depending on whether these fields are needed, and area, power and performance limitations.

## 10.2.1Types of Credits

Symphony will support two types of credits:

- ▶ Dedicated: The dedicated credits are reserved (at Target ATU) for the credit tuple. Dedicated credits are pre-allocated and available for the I-ATU to use at start up.
- ▶ Pool: Pool credits are shared between I-ATUs. Pool credits are requested by I-ATUs and cannot be pre-allocated.

## 10.2.2Types Of Credit Domains

Symphony will support two types of credit domains. More details are given in a later section.

These credit domains are:

- ▶ Credit Domain: Initiator ATUs and Target ATUs utilize credit management scheme to provide congestion feedback to the I-ATU. Traffic injection rate is controlled by the token bucket and the availability of the credit.
- ▶ Non-Credit Domain: Traffic in this domain is not regulated by credits. That is, no feedback mechanism is implemented other than link level back pressure and rate control at the Initiator.

A single I/T-ATU can be part of both domains<sup>1</sup>.

This capability allows Symphony to implement credit mechanism for some slaves and not others depending on the benefit of implementing credit management on slave and the area cost.

## 10.2.3Theory Of Operation:

There are two key components for the Credit Management scheme to work. These components are listed below:

- ▶ Credit Manager
- ▶ Rate Controller

The credit manager module is responsible for the following broad functions:

- ▶ Credit allocation to the I-ATUs.
- ▶ Processing credits at the I-ATU.
- ▶ Managing credit returns at the T-ATU.
- ▶ Processing credit requests at the T-ATU.
- ▶ Processing of credit grant messages at the T-ATU.
- ▶ Manage the credit management protocol.

The rate controller on the other hand is responsible for the following:

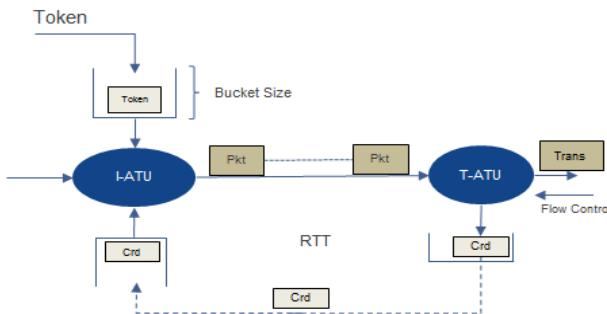
- ▶ The rate controller manages the rate at which packets are injected in to the network.  
The rate controller can be configured to regulate peak rate and burst size of the traffic.

---

1. Note: Credit and non-credit domains can co-exist. it is recommended that the T-ATU use VCs to separate the two domains.

- ▶ Rate Controller can also dynamically change the injection rate of the traffic into the network based on round trip latency. Increased round trip latency is a measure of congestion building in the system.

A high level view of ACM, is illustrated in the Figure below.



**Figure 10-2 : Different components of the ACM scheme**

Details of the token bucket rate controller and the credit manager presented in Section x.

Packets are injected in to the Fabric, if and only if, the following conditions hold true:

- ▶ Condition 1:  $(\text{Available Tokens} \times \text{Token size} \geq \text{msgSize}) \& \& (\text{Credit available})$

The above condition states that a packet will only be injected into the network if the rate controller can send the message and that the credit is available to send the message. Note that credit management operates at the Common Transport Layer.

Tokens and credits work in concert with the tokens controlling a peak injection rate in to the Fabric and the credit relaying information about the congestion in the Fabric and the slave. As we shall see in the section x, credits will also be used to regulate the ingress rate in to the Fabric.

## **10.2.4 Credit Message**

Symphony supports multiple types of credit messages. These different messages are listed below:

- ▶ Embedded Credit Req/Grant message.
    - ◆ These credit messages are embedded in the normal request/response messages.
  - ▶ Independent Req/Grant Credit Message ( $\text{Crd}_{\text{req}}$ ;  $\text{Crd}_{\text{grant}}$ ):
    - ◆ These credit messages are not embedded and can be transmitted over the regular request/response network or over a separate control network. These messages are sent when the opportunity to embed requests/grants is not available or will delay request/grant of credits.
  - ▶ Stop/Resume Credit Message ( $\text{Crd}_{\text{grant}} = 0x000$ ;  $\text{Crd}_{\text{grant}} = 1x111$ )
    - ◆ Stop message will inform the Initiator ATUs to stop transmitting to the Target ATU
    - ◆ Resume message will inform the Target ATU to start transmitting to the Source.
  - ▶ Resync Credit Message for error recovery

Credit messages can be sent using a control network if one exist or the regular Request/Response network.

The credit message fields and their definitions are as follows:

**Table 10-1 Credit Message.**

Fields	Description	Bits	Parameterizable
SrcID	Source Identifier (Initiator Identifier)	10	Yes
crMsgType	Type of Message – identifies a credit req/ grant, credit resync message	3	Yes
crMsgID	Transaction identifier	8	
crdReq/Grant	Type of request and credits req/granted	10	

#### **10.2.4.1 SrcID:**

This is ID of the ATU requesting the Credits.

#### **10.2.4.2 crMsgType:**

This field defines the type of credit message. The types are given below:

**Table 10-2 CrMsgType.**

CrMsgType[2:0]	Description
000	Stop transmitting
001	Resume Transmission (normal operation mode)
001	Credit request
011	Credit grant
1xx	Credit Resynch

#### **10.2.5 crdReq/Grant[9:0]**

This field defines the type of credit request/grant and the number of credits requested. The extended crdReq/Grant[9:0] field allows up to 256 credit requests.

**Table 10-3 Credit Req/Grant**

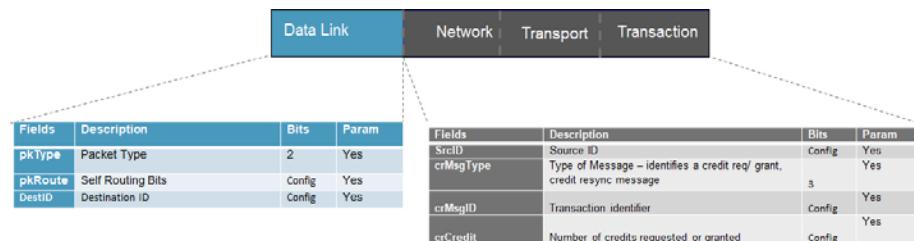
CrdReq[1:0]	Type of Credits Requested/Granted
x0	-Reserved

**Table 10-3 Credit Req/Grant**

CrdReq[1:0]	Type of Credits Requested/Granted
x1	Credits requested or granted
1x	Dedicated credits
0x	Pool credits

**Table 10-4 Credit Request Type Continued.**

CrdReq[9:2]	Number of Credit Requested/Granted
0x00-0xFF	0-255

**Figure 10-3 Independent Credit Message format**

Note that the data link packet header is same as that for regular request/response packets. The fields are parameterized and hence can be removed or reduced as needed.

If credit messages are sent over the regular network, then packet header fields in the data link section would be the same as that of a regular packet.

## 10.3 Credit Management Protocol

The credit management protocol is explained in the Figure below. Let  $d_{pi}$  and  $d_{qi}$  be the processing delay and the queueing delay at the T-ATU. Let  $crd\{u_i, w_i\}$  define the state of the credit manager at the I-ATU for the  $\{\text{SrcID}, \text{DestID}\}$  tuple, where  $u_i$  is the number of allocated credits and  $w_i$  is the number of available credits. Also let  $token\{b_i, y_i\}$  define the state of the TG, where  $b_i$  is the bucket size and  $y_i$  is the number of available tokens.

In the Figure below, the initiator has been granted 3 credits and the bucket size is 1 token only. Assume that a single token is sufficient to transmit the request or data message. Since there is one token available and at least one credit available, the condition stated in Eq 1. is true and as such a Request/Data packet can be injected in to the Fabric. Any further transmission from the I-ATU is stopped till the time a token is available. In the Figure below, once the first Req/Data packet is received at the T-ATU and the transaction is sent to the Slave, the credit is returned.

There are two options the T-ATU has for returning credits:

- ▶ Send back an independent credit message.
- ▶ Piggybacking credit message on the response packet.

The decision to piggyback the credit message or send an independent message depends on the latency that the I-ATU can tolerate. In Symphony it will be a configurable option and will be implementation dependent.

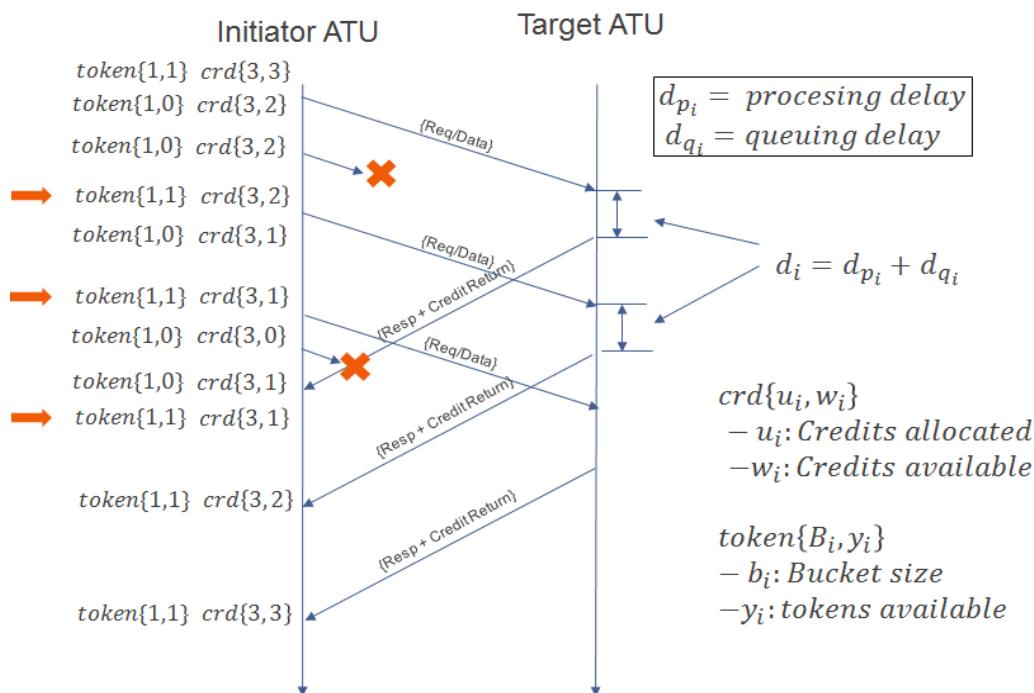
In Figure 10-5, the token bucket size is increased to 3 tokens and as such the I-ATU can send 3 back to back messages before stopping.

In Figure 10-6, the I-ATU does not have any pre-allocated credits and requests for 3 credits. The T-ATU based on the available credits, will send back a number between 1-3 credits. Once the T-ATU receives non-zero credits, it can start sending the packets. Note that once the credits are used up they are not sent back unlike the case where credits are pre-allocated.

The scheme tracks the RTT and adjusts the injection rate in to the Fabric. Section y, details enhancements to this scheme where the system will explicitly track the RTT and adjust the rate.

[discussion on tracking latency of RTT of the flow and adjusting the rate accordingly]

**Figure 10-4 Credit Management Protocol.**



**Figure 10-5 Credit Management Protocol (with credit bucket at 3)**

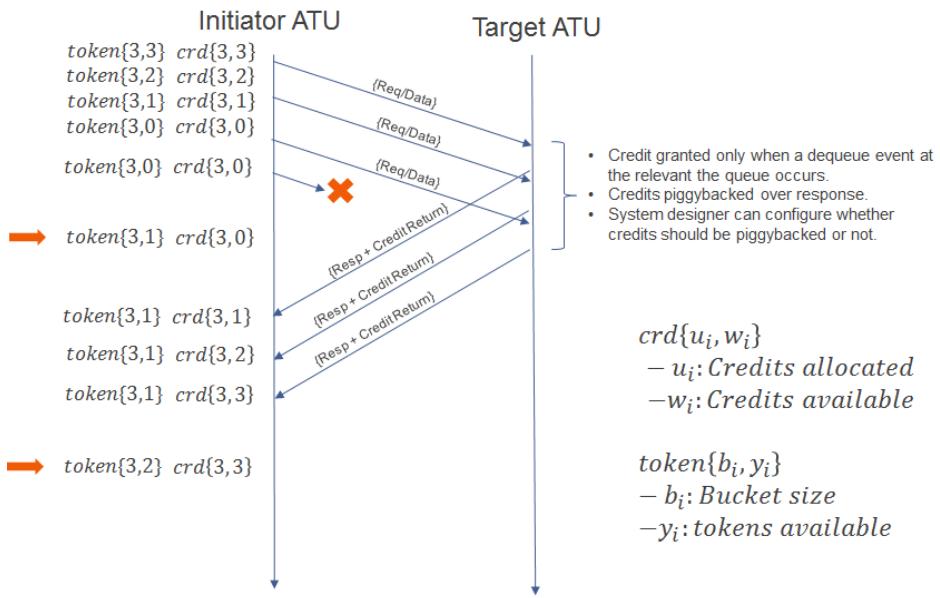
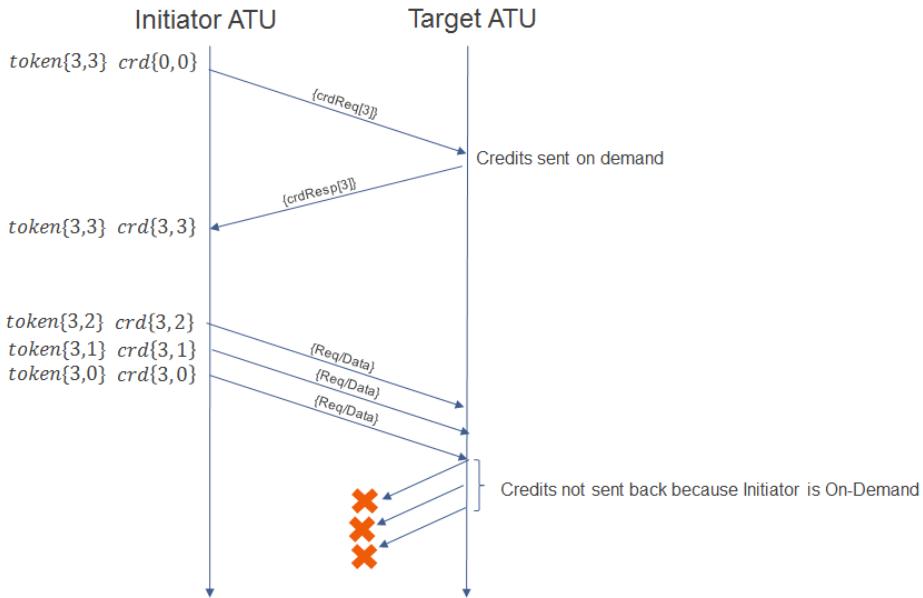


Figure 10-6 Credit Protocol (On-Demand.)



### 10.3.1 Credit Pre-allocation

The dedicated credits are pre-allocated during ATU initialization. Pre-allocation of credits depends on the following parameters:

- Let  $w_i$  be the desired injection rate in bytes/sec of flow  $f_i$  as defined by the tuple  $\{SrcID, DestID, VN, QoS\}$  at  $Src_i$ .

- ▶ Let  $w_i$  be the bandwidth at the intended Slave.
- ▶ Let also  $rtt_{ij}$  be the round trip latency for flow<sup>1</sup> traversing Src<sub>i</sub> to Dest<sub>j</sub> and  $crdSize_j$  (Bytes) be the size of each credit in bytes. Then number of pre-allocated credits  $crdPre_i$  is given by the following equation:

$$crdPre_i = rtti \times \frac{wi}{crdSize_j}$$

The above equation gives the minimum credits required to meet the rate for  $w_i$ . However, depending on the traffic profile of the I-ATU, more credits may be required. To avoid deadlock, each I-ATU that is configured to have enough dedicated to cover the max packet size configured for the I-ATU.

Credit allocation can be updated/changed at run time provided any change in credit allocation does not impact the buffer size at the T-ATU. The re-sync protocol can also be used to update the credits.

Total amount of bandwidth requested by the initiators should not exceed the sustained bandwidth of the Target.

The above equation can be used for Read requests as well.

### 10.3.2 Credit Re-sync

Credit re-sync mechanism can re-initialize the credits to recover from corrupted/lost credits. The resynch protocol can also be used to reconfigure dedicated credits. The protocol is defined as follows:

- ▶ The T-ATU initiates a resynch process by sending a re-synch message to the I-ATU/I-ATUs.
- ▶ After sending the re-synch message the T-ATU stops sending credits back to the I-ATU.
- ▶ Upon receiving the re-synch message the I-ATU will perform the following steps:
  - ◆ Stops accepting all traffic from the Master.
  - ◆ Sends already the buffered transactions.
  - ◆ I-ATU transmits the resynch packet back to the T-ATU. This resynch message serves to flush the system of packets in the system.
- ▶ Upon receiving the resynch message from the I-ATU, the T-ATU responds with credit grant message carrying the new credit allocation.

## 10.4 Token Generator (TG):

The TG generates the token at the specified rate. TG is specified by the following parameters:

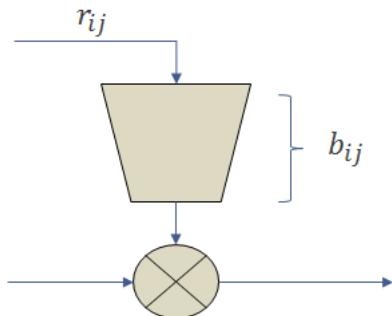
- ▶ Rate at which the Master  $i$  intends to send traffic to target  $j$ . This rate is translated in to the rate at which tokens will be generated. The token generation rate will be referred to hereon as  $r_{ij}$ .

- 
1. RTTi is the average round trip latency for the flow. This latency number can be obtained via simulations or other means which are reasonable. Note that this value can change as the design goes through different stages of the flow. Sensitivity analysis should be done, before configuring the RTT and credits

- ▶ Data burst size that Master  $i$  intends to send to target  $j$ , from here on referred to as  $C_{ij}$ .

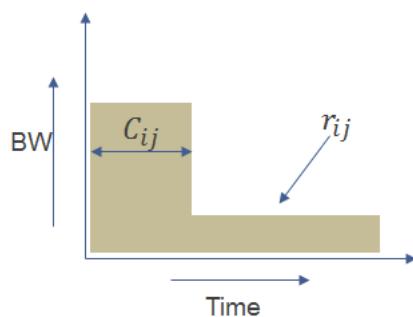
TG acts like token bucket and as long as, enough tokens are available, a msg can be sent in to the fabric provided enough credits are also available. Note that the TG is a standalone module and can be used without credits.

The rate  $r_{ij}$  determines the rate at which the tokens will be added to the counter. The counter can accumulate up to  $b_{ij}$  tokens. Once the counter reaches this limit, it will not accumulate any more tokens. The Figure below illustrates the operation of the Token Generator.

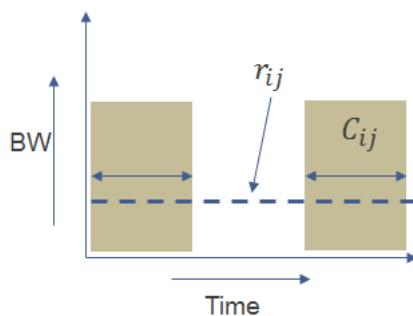


**Figure 10-7 TG with token rate  $r_{ij}$  and burst depth  $b_{ij}$ .**

The above TG will enable the following traffic profiles, as illustrated in the Figures below, for the Master.



**Figure 10-8 Traffic profile for TG with rate  $r_{ij}$  and burst size  $C_{ij}$ .**



**Figure 10-9 Traffic profile for TG with rate  $r_{ij}$  and burst size  $C_{ij}$ .**

The following equation governs the relationship between  $C_{ij}$ ,  $r_{ij}$  and  $b_{ij}$

- ▶ Let  $L_i$  be rate of the interface that the I-ATU has with the Fabric.
- ▶ Then:  $C_{ij} = b_{ij} + r_{ij}/L_i$

The maximum tokens that can be accumulated is determined by the burst size required by the Initiator. Based on the traffic profile of the I-ATU,  $r_{ij}$  and  $b_{ij}$  will be determined.

Note:  $C_{ij}$  should be configured so that it is at least greater than the max packet size.

Note that, the availability of the token and the availability of the credit is not synchronized. That is, there may be a case when tokens are available but no credit visa versa.

Slow speed I-ATUs will generally have  $r_{ij}$  that are a multiple of the RTT and as such the credits would return back to I-ATU, under normal conditions, before the token is generated. On the other hand, for high speed I-ATUs the credits may return after the token is generated.

In order to generate tokens, the I-ATU will use the local clock. Note that tokens are always generated for following tuple:  $\{SrcID, DestID, VN, QoS, read/write\}$ .

Write request and write data are accounted for together. That is the TG for writes is configured to take into account the write request and data together.

A read request will only be allowed to proceed, if there were enough tokens to cover the read response size. For a read request to proceed the following have hold true:

- ▶ Available Tokens  $\geq$  Read Response size

The reason for this rate limiting is that Symphony does not employ end-to-end sustained congestion controls on the response path. Symphony will employ transient congestion control mechanism on the response path.

For I-ATUs that are configured to request tokens on demand, TG can be used to trigger the ATU to send the credit request to the target. In this case the TG simply acts as a timer along with generating tokens at the configured rate.

The following condition has to hold, for the Initiator to request a credit:

- ▶ There are no outstanding credits requests for the tuple  $\{SrcID, DestID, VN, QoS, Request/Data\}$

TG uses the local I-ATU clock to generate tokens. No global synchronization is required.

## 10.5 Credit Manager

The credit manager is implemented in the CT layer of the ATU (initiator or target).

### 10.5.1 Types of Initiators

- ▶ High-Touch Initiators
  - ♦ Low Latency such as Processor Core
  - ♦ Real-time such as GPUs, high speed IO peripherals (GE)
  - ♦ High bandwidth such as different IP blocks (SEC, Pattern Matching Engine, ...)
- ▶ Low-Touch Initiators

- ◆ Low touch initiators are those which do best effort and require low bandwidth and are not latency critical. Although dedicated and Pool credits can be assigned to any {SrcID, DestID, VN, QoS}, it is recommended that dedicated credits be assigned to low latency and guaranteed bandwidth, or high bandwidth requesting Initiators.

The number of outstanding credits directly impacts the buffering requirement at the T-ATU.

Pool credits are not pre-allocated but assigned on demand. That is, the initiator needs to send a shared credit request to the target ATU.

## 10.5.2 Initiator ATU Credit Manager

Although the ACM utilizes both the rate limiter and the credit manager to implement the credit management scheme, the credit manager is architected to be a standalone module.

The credit manager operates at the message in the network layer.

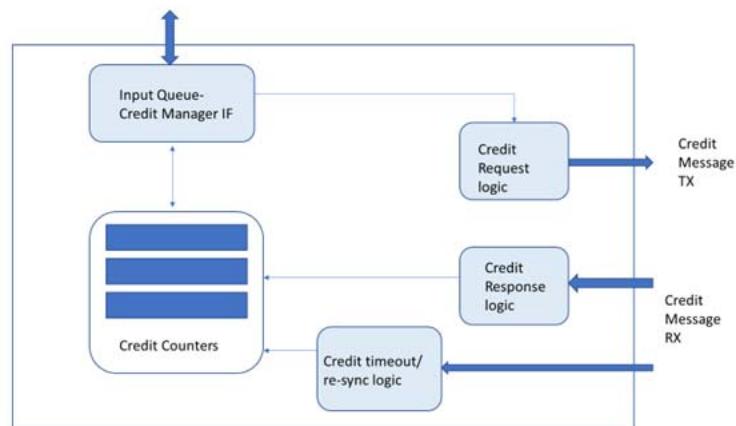
## 10.5.3 CT Layer

Credits are allocated to the {SrcID, DestID, VN, QoS, Req/Data} tuple.

The credit manager consists of the following modules as shown in Figure :

- ▶ Credit Response module
- ▶ Credit Request module
- ▶ Credit update module
- ▶ Resynch module

**Figure 10-10 I-ATU Credit Manager module**



Credits can be received in two ways: Credits embedded in the response messages; and/or via independent credit messages. The Credit response unit has to process the credits received along with the flow information that the credit belongs to. The credits then are added to the appropriate flow.

The credit counter logic updates the credit counters based on request to add or subtract credits from the counter. The counters are identified by the tuple as defined by {DestId, VN, QoS}. Fields within the tuple can be masked.

The credit timeout (“Credit Timeout” on page 165) and re-synch module is responsible for managing credit timeout events and the re-synch protocol.

The credit request module is responsible for requesting credits from the T-ATU. The credit request module can be configured to request credit if credits are not pre-allocated to the I-ATU. Requests can be generated periodically triggered by a certain configured clock rate or if there is a transaction waiting for credits.

The request for credits should be enough to match the max packet size configured for that I-ATU.

It is possible to enhance the functionality of the credit scheme to allow I-ATUs that are allotted dedicated credits to also request Pool credits. Pool credits once delivered should be used before any dedicated credits are used. The reason for this, is that the Pool credits should be returned to the associated T-ATU as soon as possible. If this enhancement is used, the timeout logic would also conduct unused Pool credit collection and return them to the T-ATU.

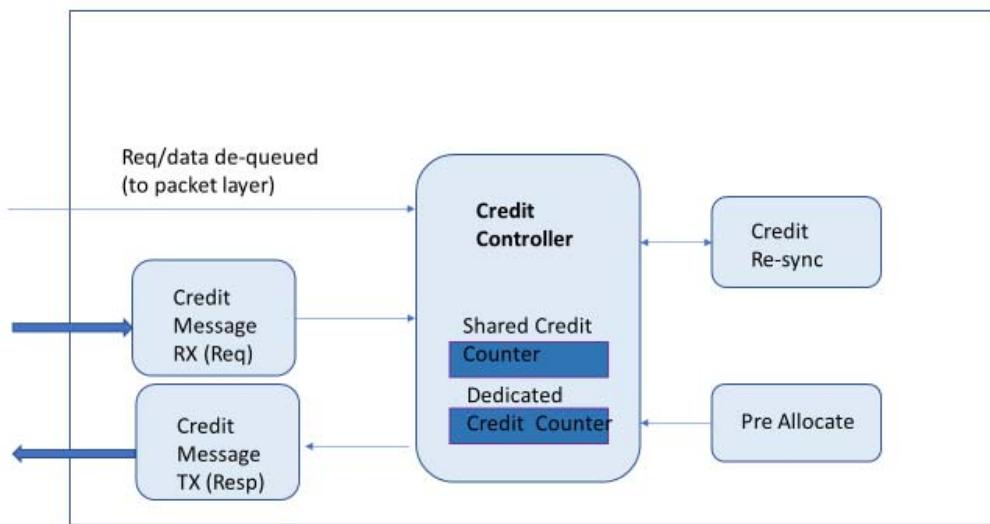
## 10.5.4 Target ATU Credit Manager

The Credit manager module at the Target ATU (in CT layer) is composed of multiple modules as shown in [Figure 10-11](#). The Credit resynch module is responsible for sending the re-synch message to the I-ATU in case resynch is required. It is also responsible for managing the re-synch protocol.

The credit controller keeps the state for Pool credits in terms of the number of available credits. For dedicated credits, in the simple implementation, does not need to keep track of any credit exchange with the I-ATU. The credit control module simply waits for the transaction to dequeue to be sent to the slave, and returns the credit back to the I-ATU.

The credit controller has to decide whether to send the credits as independent message or piggy back it on the regular response packet to the I-ATU. This is a configuration parameter. The configuration will determine the following:

- ▶ Always send an independent credit message.
- ▶ Always piggy back the credit message on the response packet to the same tuple
- ▶ Return the credit via the regular response message. If the configured timer expires, then send the message as an independent credit message.

**Figure 10-11 Credit Manager (Target ATU)**

## 10.6 Credit state and algorithms

### 10.6.1 Credit state variables

- ▶ The T-ATU only manages state for the Pool credits. In the simple implementation it does not monitor

### 10.6.2 Credit Generation

Credit is generated when one of following events happen:

- ▶ Request for credits is received –
  - ◆ A credit request can be sent for both dedicated and Pool credits. Dedicated credits will normally be pre-allocated at initialization time. A credit request for Pool credits can be granted (ACK) or rejected (NACK). A credit grant message with zero grants is a NACK.
  - ◆ Enhancement to the basic scheme is that the I-ATUs that have dedicate credits can also request for shared credits. Symphony provides a configurable threshold which will prevent distribution of pool credits to I-ATUs that are allocated dedicated credits, if the available number of Pool credits is below that threshold. This is to prevent starving those ATUs that do not have dedicated credits and depend on Pool credits for sending transactions into the Fabric.

- ▶ A dequeue event happened:
  - ◆ The dequeue event is the result of a transaction leaving the T-ATU and transmitted to the Slave. If the dequeued transaction was associated with a dedicated credit then the credit is returned back to the I-ATU that initiated the transaction. Otherwise, if the credit associated was a pool credit, it is returned to the pool and not sent back.

### 10.6.3 Credit Return Algorithm

The packet header contains the single bit to indicate whether the packet/Transaction has an associated dedicated credit or pool credit or no credit.

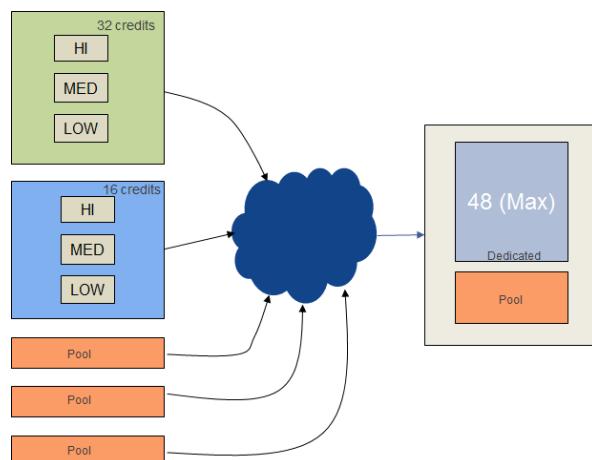
Dedicated credits: Dedicated credits are returned (grant) back to the I-ATU, by sending a credit message, or piggybacking the credit grant on a response message back to the I-ATU.

Pool credits: Pool credits are returned to the shared credit pool of the credit manager (in the Target ATU), which is stored in a credit counter (for shared credits only). If Pool credits can not be granted, a credit message is sent with zero credits back to the requesting I-ATU.

No credits (Non-Credit Domain): No credits to be returned.

## 10.7 Credit Mechanism Example

**Figure 10-12 Credit Pools Example**



The example above shows 5 initiator ATUs with 1 Target ATU. The Target ATU has 64 credits, with 48 marked as dedicated and 16 shared. The dedicated credits are divided among 2 high-touch initiators – 32, 16 respectively. The low-touch initiators have no dedicated credits, and request Pool credits as needed. One of the high-touch initiators has both dedicated (16) and Pool credits, and it can request for Pool credits if it runs out of the dedicated credits. The Target ATU says has a threshold of 8 for Pool credits, below which only credit requests from I-ATUs that are not granted dedicated credits will be honored.

## 10.8 Credit Domains and Virtual Channels

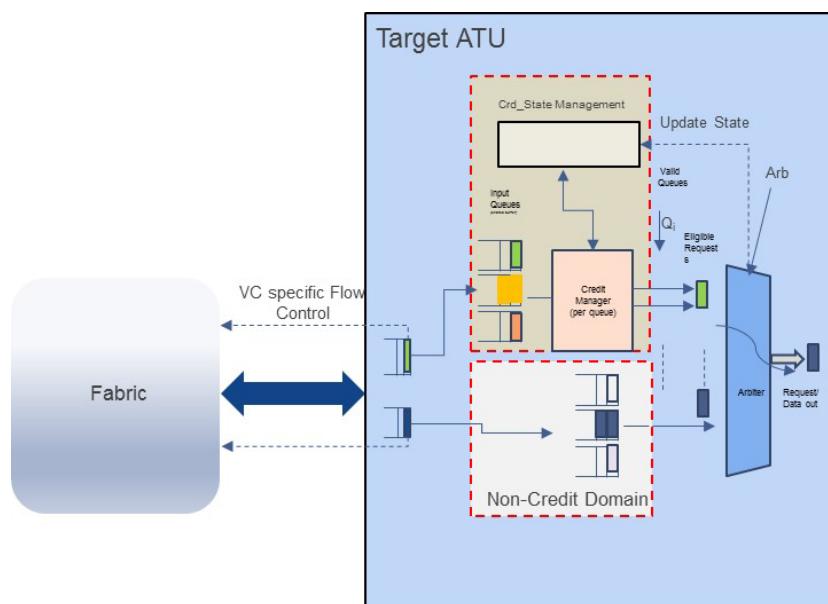
The credit scheme allows both Credit and Non-Credit domain to exist in the same Fabric/Interconnect. An ATU can participate in both domains.

The ATUs segregate Credit Domain traffic from Non-Credit Domain traffic by using per-domain Virtual Channels at the I-ATU, T-ATU and the Switches.

The T-ATU will issue domain specific link level flow control using the virtual channels. Domain specific virtual channels at the Target ATU can simply be a domain specific pipeline stage, however, the interconnect should be appropriately setup using FT and PL switches as well as the appropriate adapters to separate Domain specific traffic in the Fabric.

The choice of using VCs to segregate traffic or to use a different physical network would depend on the topology and the cost tradeoffs between a unified VC based Fabric or two separate physical fabrics.

**Figure 10-13 Credit Domains (Target ATU)**



## 10.9 Credit Buffer Sizing

ACM scheme decouples the allocation of credits from the buffers that are allocated. ACM does not require that a buffer should be allocated for each outstanding credit. Rather the buffers are sized by taking into account the drain rate at the Target ATU and the probability of back pressuring the link in event of congestion.

We define the following:

- ▶ Let  $\lambda_{ij}$  be the injection rate between  $I\text{-ATU}_i$  and  $T\text{-ATU}_j$ .
- ▶ Let  $\Lambda_j$  be defined as ingress rate at  $T\text{-ATU}_j$ , where  $N$  is the number of initiators sending traffic to  $T\text{-ATU}_j$ :

$$\diamond \quad \Lambda_j = \sum_{n=1}^N \lambda_{ij}$$

- ▶ Let  $\mu_j$  be the egress rate from the  $I\text{-ATU}_j$  to the Slave.
- ▶ Let  $p_i$  be the number of pre-allocated tokens for  $I\text{-ATU}_i$
- ▶ Let  $s$  be the total number tokens allocated by  $T\text{-ATU}_j$ :

$$\diamond \quad s_j = \sum_{i=1}^N p_i$$

- ▶ Let  $C_j$  be the buffer size at the  $T\text{-ATU}_j$ . Then  $C$  is given as:

$$\diamond \quad C_j = \min \left[ 1, \left( 1 - \frac{\Lambda_j}{\mu_j} \right) \right] \times s$$

The buffer size can be further reduced by using the M/M/1/K queueing analysis to determine the probability of asserting link level flow control.

## 10.10 Credit Timeout

Symphony can be configured to have a timeout counter to make sure that the credit request is responded to within a configured time.

For shared credits, a shared credits collector timeout can be implemented to return unused shared credits.

## 10.11 Network Latency and RTT Calculation

The time taken by a transaction to traverse from the I-ATU to the T-ATU and back is called the round trip time (RTT). The RTT is composed of the following terms:

- ▶ Let:
  - ◆  $L_{forward}$  be the latency of the Request path, such that:
    - ✓  $L_{forward} = d_{I\text{-ATU}} + d_{T\text{-ATU}} + (N) * d_{NE} + (N+1) * d_{link}$ ; where  $d_{I\text{-ATU}}$  is the delay experienced by the packet in the I-ATU;  $d_{T\text{-ATU}}$  is the delay experienced by the packet in the T-ATU;  $d_{NE}$  is the delay experienced by the packet in network elements such as switches, rate adapters, width adapters, etc; and  $d_{link}$  is the transmission delay.
  - ◆  $L_{return}$  be the latency of the credit return path. Note the credit return path latency could either be that of independent credit messages or of embedded credit messages.
    - ✓  $L_{return} = d_{I\text{-ATU}} + d_{T\text{-ATU}} + (N) * d_{NE} + (N+1) * d_{link}$

Hence the round trip time is as follows:

$$\triangleright \quad RTT = L_{forward} + L_{return}$$

Each of the terms within the forward and return path latency can be further decomposed as follows:

- ▶  $d_{I/T-ATU} = d_{processing} + d_{queueing}$
- ▶  $d_{NE} = d_{processing} + d_{queueing} + d_{arb}$

As congestion inside the network or at the Slave builds up the  $d_{queueing}$  portion of the delay increases and it is precisely this portion of the delay that causes network jitter.

RTT calculation is the sum of averages of each of the two component delays. RTT is not based on zero load delay. RTT is the estimated value.

Since RTT is used to size the buffers at the T-ATU, the weighted average RTT over all paths should be used to size the buffers as described in see “[Credit Buffer Sizing](#)” on page 164.

## 10.12 ACM Enhancement:

The ACM scheme described so far, implicitly adjusts the injection rate from the I-ATU in to the fabric. This section describes an enhancement to the basic scheme where the I-ATUs explicitly adjust the injection rate by modifying the rate of the Token Generator.

The I-ATU timestamps the packets as they enter the I-ATU VCs if they are present and the Fabric if they are not. At the corresponding T-ATU the time stamp is copied and then sent back either through independent credit message or piggybacking it on the regular response message.

When the packet reaches the I-ATU, the embedded time stamp is compared against the current time at the I-ATU. The time difference between the time stamp and the current time will determine the *RTT* at the time the packet was sent.

The I-ATU compares these *RTT* to determine if the injection rate needs to increased or decreased. An increase in *RTT* indicates congestion buildup either within the network or at the Slave and a reduction in *RTT* indicates reduction in congestion in the network.

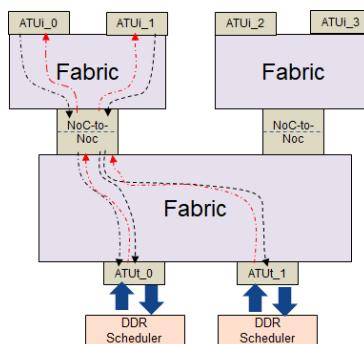
The basic idea of this enhancement is to explicitly change the injection rate via the Token Generator based on the feedback received through variations in *RTT*.

An average RTT can be calculated using a moving window averaging over the past “n” values to create a low pass filter effect in order to reduce hysteresis.

## 10.13 Hierarchical Credit Management

This section will describe how the credit management scheme described in the previous sections is extended to construct hierarchies.

The credit management function is the responsibility of the CTL Layer in both the Initiator and the Target ATUs.



**Figure 10-14 Hierarchical Credit Management: The NoC-to-NoC adapter terminates and initiates credits for upstream Fabric and the downstream Fabric. Credit return is represented by the red arrows**

In an hierarchical system as the one shown in [Figure 10-14](#), the NoC-to-NoC adapter terminates the credits from the upstream Fabric and returns them to the appropriate ATUi. While at the same time, initiating credits for the downstream Fabric. The credits could be allocated to fully utilize the Slave socket bandwidth or the interface bandwidth at the south end of the NoC-to-NoC adapter.

The NoC-to-NoC adapter is shown in [Figure 10-16](#). Note only functionality relevant to Credit Management is shown in the diagram. The adapter is split into CTLt and CTLi layers. The CTLt layer receives the packets from the upstream Fabric via the Virtual Channels. The packets are put into the appropriate flow queue based on a set of criteria. The logic in the CTLt layer can match any field in the transaction to map it to a particular Flow Queue. Packets are flow controlled via VCs if VCs are used in the network.

Once the packets are put in the appropriate queues, the queues are arbitrated on by the appropriate scheduling algorithm (one of the Symphony Schedulers). The flow queue selected is serviced and packet is transferred from the CTLt to the CTLi. On the CTLi side, the packet is then remapped to a flow queue. This remapping may be different from that done on the CTLt side. Sometimes it may be required to group all, e.g. priority flows into a single Flow Queue on the CTLi side, while leaving them separated on the CTLt side.

The flow queues are serviced and the packets are put in the appropriate VCs as dictated by the mapping.

The credits will be allocated to the flows as a set of two. One between the ATUi and the CTLt and other between the CTLi and ATUt at the downstream Fabric.

Multiple flows can have different type of QoS requirements. The CTLi and CTLt layers cater to scenario where some flows would be high priority flows and other lower priority. Each flow queue in the CTL layer can represent a priority if the system so configured.

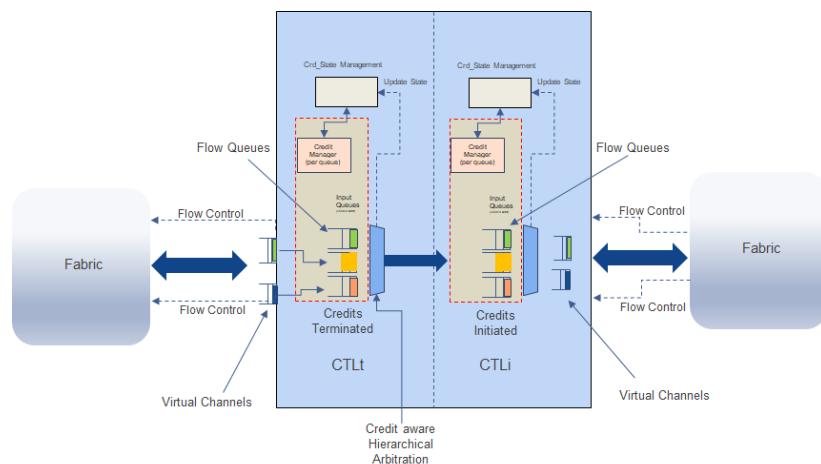
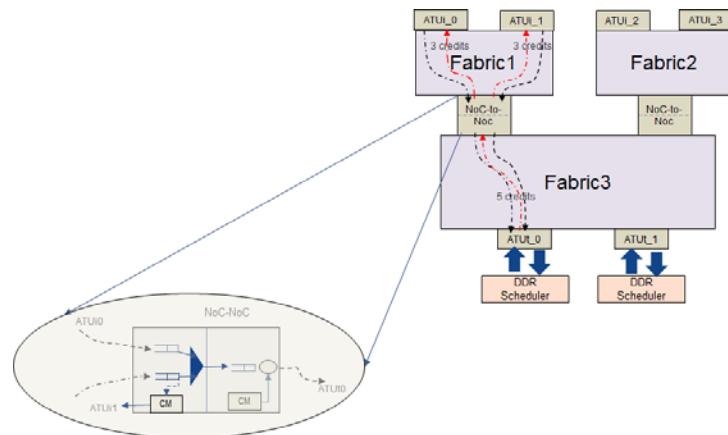
Note: Total buffering required in the NoC-to-NoC adapter should need not exceed the total outstanding credits. It does not matter which blocks contains the most buffering.

Consider the example shown in [Figure 10-15](#). The network is composed of multiple core clusters. Two initiators ATUi\_0 and ATUi\_1, which are in the same cluster, are sending transactions to ATUt\_0. The Fabrics are connected to one another via the NoC-to-NoC adapter. Let us Assume that it requires 3 credits to fully utilize the link bandwidth for each initiator from the ATUi to the NoC-to-NoC adapter and another 5 credits to fully utilize the link bandwidth from the NoC-to-NoC adapter to the DDR. In the network shown in the example, each of the ATUi would be allocated 3 credits to enable them to fully utilize the link bandwidth if the other ATUi is not using it and 5 credits would be allocated to the NoC-to-NoC adapter. The NoC-to-NoC adapter buffering the worst case would be equivalent to 6 credits worth of data. The NoC-to-NoC adapter can be configured to split the bandwidth to the DDR in any ratio, or assign higher priority to one flow and lower to the other while maintaining starvation protection.

Packets from the upstream fabric are queued (cut through) in the appropriate flow queues. The flow queues are then arbitrated based on how the system wants to treat the two flows. The arbiter is a ready/credit aware arbiter. While the two flows could be queued in different queues in the CTLt, they can be

mapped to the same flow queue on the CTLt side as shown in [Figure 10-15](#). Each flow queue is credit controlled. This flexibility allows different mapping arrangements and hence the system as a whole can offer multiple levels of QoS.

**Figure 10-15 Example of Hierarchical Credit Management**



**Figure 10-16 : NoC-to-NoC adapter - functions relevant to credit management: Flow queues, VCs and Credit manager**

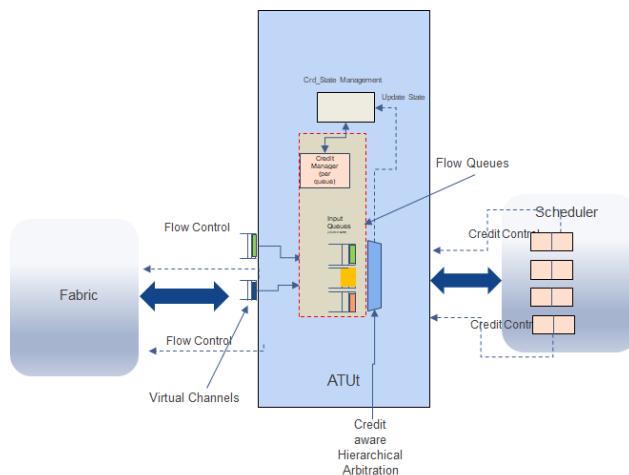
## 10.14 Credit Management and Memory Controller Interaction

The credit management scheme detailed in this chapter can be easily extended to interface with a memory scheduler provided the scheduler is able to provide information on the availability of buffer space in the memory controller. This information can be provided to the ATU via user bits as an example or a well defined interface.

If user bits are used, they would inform the ATUt of how many credits/buffer granules are available for a particular queue in the scheduler. Based on the buffer availability and the information about the QoS of each associated queue the ATUt will forward the command and data to the scheduler.

The ATUt will use a credit-aware arbiter to make sure that only those queues are eligible for scheduling that have a credit available at the other end. Figure below illustrates how the scheme will work.

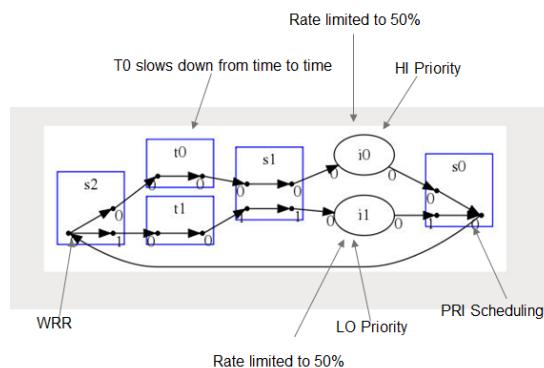
**Figure 10-17 ATUt and the memory scheduler**



## 10.15 Performance Results

In order to investigate the performance of the system when credits are enabled, the Symphony performance model was used. Two network scenarios were used. The network topologies are illustrated in [Figure 10-18](#) and [Figure 10-22](#):

**Figure 10-18 Network Scenario 1: Two initiators and two targets**



Network Scenario 1, contains two initiators and two target connected through a simple network with two switches in the request network and a single switch in the response network. Also note that all throughput numbers are round trip numbers.

The following describe the simulation setup:

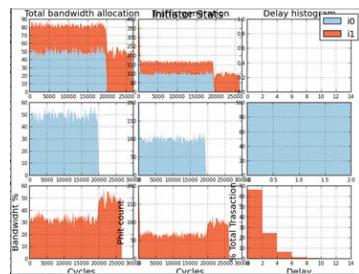
- ▶ Number of Flows: 2
  - a. Flow 1: I0 → T0; High Priority

b. Flow 2: I1 → T1; Low Priority

- ▶ Both Initiators are rate controlled to 50%
- ▶ a. Initiator 1 further rate controlled to 30% by credits
- ▶ Initiator traffic is generated using a uniform distribution.
- ▶ Fixed size packet are used. A single credit corresponds to the packet.
- ▶ Switch S0 is configured to arbitrate based on PRI scheduling
- ▶ Switch S1 ports are configured to have WRR arbitration.

The [Figure 10-19](#) below shows the results of the simulation based on the above configuration:

**Figure 10-19 Scenario 1 results: Initiator 0 acquires 50% of the bandwidth while Initiator 1 only gets 30%.**

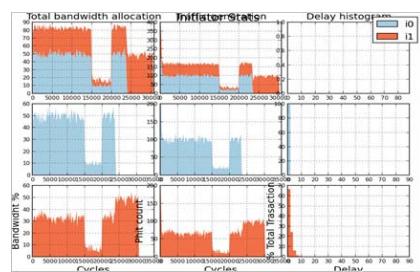


In above figure, Initiator 1 is rate limited to 30% by reducing the amount of credits given to the Initiator. Remember, the Initiator can only transmit if it has a credit.

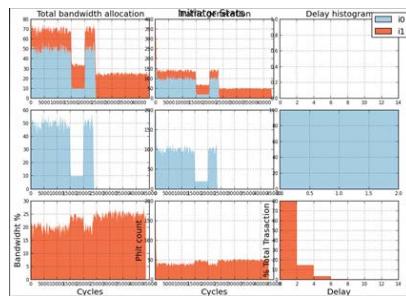
In [Figure 10-20](#), Target 0 slows down by 90% and as a result throughput of Initiator 0 falls to 10%. Notice that throughput of Initiator 1 also falls down to 10%. The reason is that both Initiator 0 and Initiator 1 share a common link and traffic from Initiator 0 has higher priority. The reason why Initiator 1 is able to sneak some packet through is because the way the rate limiter works in the simulation. The rate limiter maintains an inter-packet delay specified in the simulation. For example, if the rate is limited to 50% of the link rate, then the rate limiter will make sure that Initiator sends a packet every other packet time. The rate limiter starts to count the inter-packet delay only when the earlier packet leaves the ATU. Hence when Initiator 0 slows down to 10% due to the target 0 slowing down, the rate limiter at Initiator 0 holds packets for one packet time. It is this inter packet gap, that is utilized by Initiator 1 to send the packet out. That is why Initiator 1 is able to send packets every other packet time.

In [Figure 10-21](#), we show the results when credits are enabled on Initiator 0. The credits limit the injection rate of Initiator 0 and lets Initiator 1 reclaim the lost bandwidth.

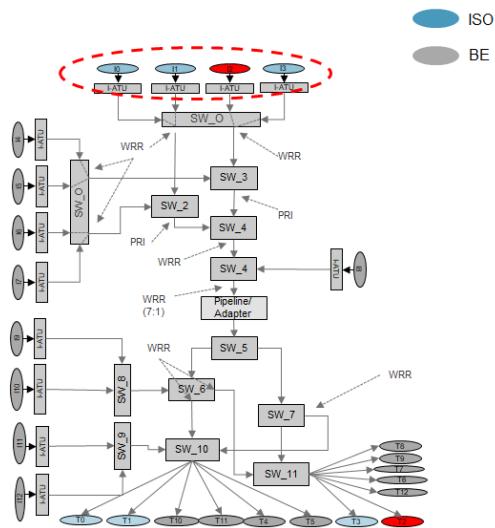
**Figure 10-20 Scenario 1 results: Target 0 slows down by 90% and reduces Initiator 0's throughput to 10%.**



**Figure 10-21 Scenario 1 results: Credits enabled on Initiator 0, allow Initiator 1 to regain lost bandwidth**



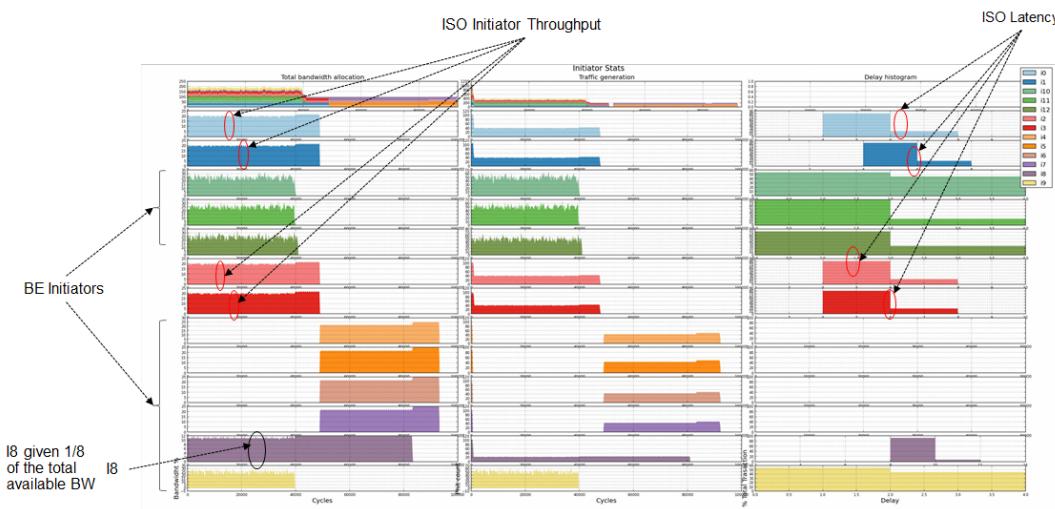
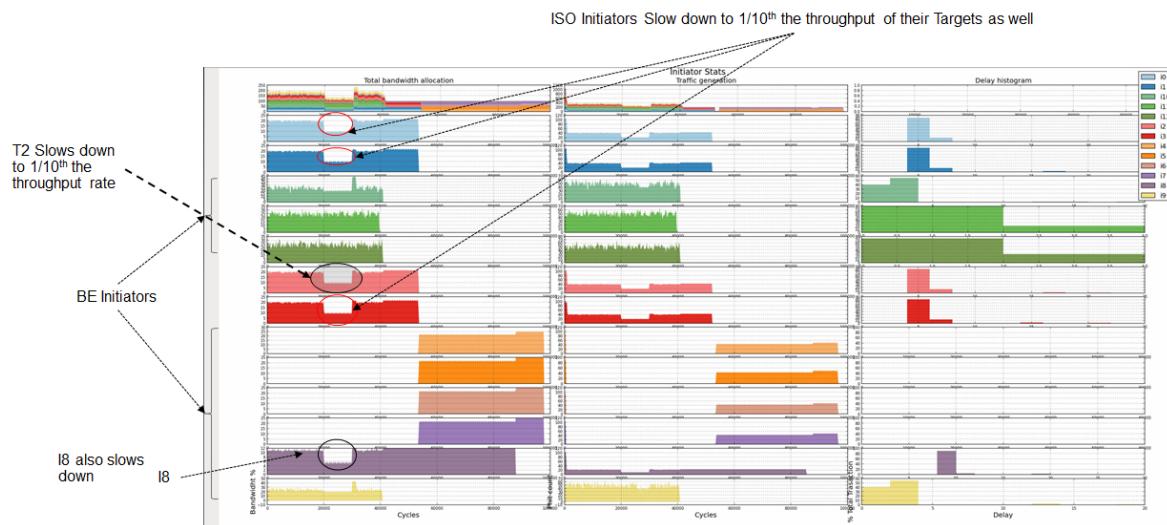
**Figure 10-22 Network Scenario 2: Multiple Initiators and Multiple Targets**



The network topology shown above was also simulated. The simulation configuration was as follows:

- ▶ Initiators I0, I1, I2, I3 are all high priority real-time sources of traffic
- ▶ All other initiators are best effort traffic
- ▶ Initiators I0-I8 are rate controlled at 50% of the link rate
- ▶ Initiators I9-I12 are rate controlled at 25% of the link rate.
- ▶ Each initiator sends to its corresponding target only.
- ▶ Target 2 slows down for some period of time. This slow down could represent congestion in the downstream subsystem connected to target 2.

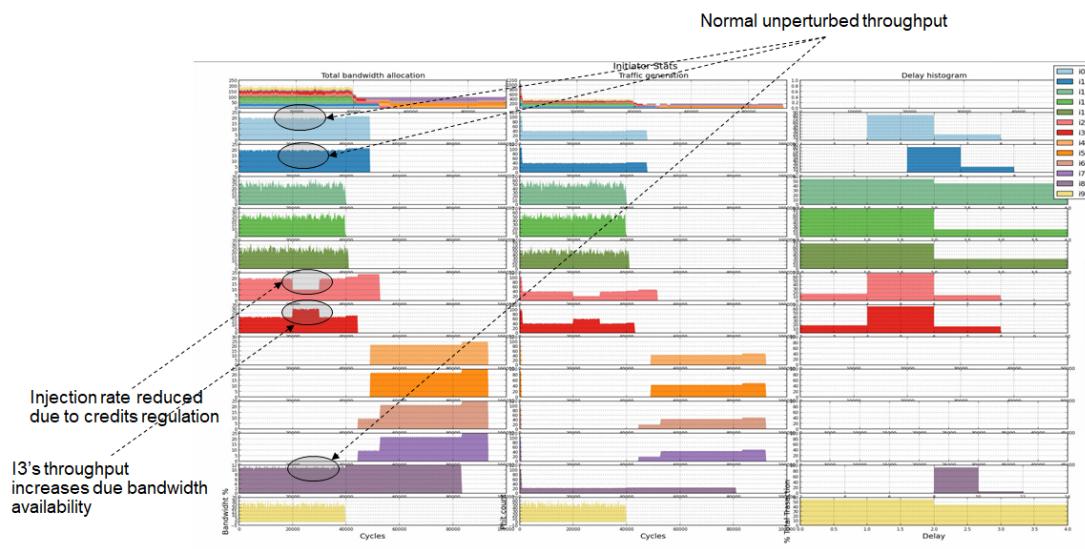
The rate limiter in this scenario are of the same type as in Scenario 1. Under normal conditions the results show that the ISO initiators evenly divide the bandwidth of the link between them while initiators 4, 5, 6 and 7 do not receive any bandwidth as shown in [Figure 10-23](#).

**Figure 10-23 Scenario 2: Result under normal conditions****Figure 10-24 Scenario 2: Target 2 takes 10x more time to process packets than before.**

In the figure above, Target 2 is configured to take 10x more time to process packets for a small period of time. This processing delay could represent congestion in the subsystem attached to target 2. The impact of this slow down, due to the flow through and wormhole routing, impacts not only Initiator 2 but the other initiators that share the link. This could be seen by the dip in the throughput graph of these initiators.

As soon as credits are enabled on Initiator 2, the traffic injection rate at Initiator 2 is adjusted based on the target 2 processing rate and the other Initiators regain their fair share back.

Note: all switches used were Flow-Through switches.

**Figure 10-25 Scenario 2: Credits enabled on Initiator 2**



# Scheduling Algorithms

## 11.1 Architectural Purpose and Requirements

The purpose of defining multiple scheduling algorithms is to provide Symphony with the necessary set of arbitration algorithms that will enable Symphony to meet the QoS requirements of different traffic classes as defined in this document.

The range of algorithms varies from simple to complex. Appropriate algorithms will be chosen at the arbiters to meet the QoS requirement as well as meet timing/speed and area constraints.

The algorithms meet two key requirements for the scheduling algorithms:

- ▶ Algorithms are implementable.
- ▶ State space required is by the algorithms is reasonable.

## 11.2 Traffic Classes:

Traffic within an SoC is segmented in to six distinct categories. These categories may or may not simultaneously exist in the SoC.

These categories are:

- ▶ Network IO Traffic -- **Bursty with bandwidth requirements and latency limits.**
  - ◆ Ethernet traffic needs to be buffered as soon as it enters the SoC. Otherwise buffer overflow can happen on the connected Ethernet link.
- ▶ Traffic generated by the application running on the host processor – **Non-critical Bursty but low latency.**
  - ◆ Application stalls due to latency in the processor subsystem lowers the overall system performance
- ▶ Traffic generated by mission critical components – **Strictly real-time.**
  - ◆ Airbag systems, auto pilot systems in cars, pace makers
- ▶ Traffic generated by devices that is **Isochronous** (periodic).
  - ◆ Video display traffic.
- ▶ Low speed Peripheral IO – **Low bandwidth and looser latency limits.**
  - ◆ I<sup>2</sup>C, UART, ..
- ▶ Traffic generated as a result of interacting with Hardware accelerators – **Bandwidth Guarantees.**

- ♦ Security engine.
- ♦ Compression/decompression engine

## 11.3 Scheduling Algorithms:

### 11.3.1 Eligibility Condition:

Eligibility conditions may vary depending on local conditions. In Switches, the port becomes eligible if it has a valid PHIT to send. In the ATU, the eligibility condition for a queue may include whether the queue has the credit in addition to valid PHIT available for transmission.

### 11.3.2 Round Robin:

Eligibility:

The Input Port/VC is eligible if it is backlogged, that is, it has a valid PHIT to send.

Behavior:

The algorithm will service the next eligible input Port/VC.

Algorithm:

```

. Let  $N = \text{number of input Ports}$ 
. Let the last port serviced =  $\text{lastService}$ 
. for( $i = 0; i < N; i + +$ ){
    . if( $i = \text{!lastServiced} \&\& \text{inPort}[i] == \text{eligible}$ ){
        . Service  $\text{inPort}[i]$ 
        .  $\text{lastServiced} =$ 
        .  $i$ 
    }
}

```

LastServiced value is reinitialized once all the ports are visited.

### 11.3.3 Priority Scheduling:

Eligibility:

An Input Port/VC is eligible if it has a valid PHIT to send

Behavior:

The algorithm services the highest priority Input Port/VC as long as it has a valid PHIT to send.

Algorithm:

```

.   Let  $p_i$  be the priority assigned to  $inPort_j$  or  $VC_j$ 

.   Let  $p_i > p_{i+1}$ 

.   for ( $j = 0 ; j < N; j + +$ ){

.     /* Sort the ports in order of priority */

.     If(  $inPort_j \rightarrow p_i > \forall inPort_i \rightarrow p_i$  )

.       While(  $inPort_j$  is eligible)

.         a. service  $inPort_j$ 

.         break;

}

}

```

### 11.3.4 Weighted Round Robin (WRR):

Eligibility:

A port is eligible for arbitration if it has a PHIT available and if the Weight associated with the port is greater than zero.

Behavior:

The algorithm allocates weights in accordance with the bandwidth allocated to each input port. The unit of weight corresponds to a single packet. The algorithm replenishes the weights when there is no port that can be serviced because either it does not have a valid PHIT or the weight is less than zero.

Algorithm:

Let:

```

.    $N$  = number of ports

.    $t_i = w_i$  = weight assigned to  $inPort_i$ 

.    $W =$ 

.

.    $portServiceable_i =$  port  $i$  is serviceable iff it has a valid PHIT  $\&\& w_i > 0$ 

.    $b_i =$  bandwidth allocation of input port  $i = \frac{w_i}{W} \times (\text{Output port bandwidth})$ 

```

- .  $lastServiced = \text{last input port serviced by the scheduler}$
- .  $\text{While(at least one portServiceable\_i == true)} \{\text{if (portServiceable}_{lastServiced}\}$
- . {
  - i.  $\text{service inPort}_i$  (service PHITs until the last PHIT)
  - ii.  $w_i = w_i - 1;$  }
  - .  $lastServiced = (i + 1)modN$
  - $w_i = t_i\}$

### 11.3.4.1 States:

Below is state table for a PHIT based WRR arbiter.

$$input_i = \{0 \text{ if there is not valid PHIT; } 1 \text{ if there is a valid PHIT}\}$$

$$weight_i = \{0 \text{ if weight} = 0; 1 \text{ if weight} > 0\}$$

Note: In a packet based arbiter once the input wins arbitration it will lock the input and will not arbitrate until the last PHIT from the locked input is serviced. Arbitration only happens on First PHITs.

States	Input	Weight
S0	0	0
S1	0	1
S2	1	0
S3	1	1

Replenish weights if none of the VCs are in state S3. The arbiter cannot make forward progress in states S0, S1 and S2. Decrement counter once the PHIT is serviced.

State table for packet based WRR arbiter:

$$input_i = \{0 \text{ if there is no first PHIT; } 1 \text{ if there is a valid first PHIT}\}$$

$$weight_i = \{0 \text{ if weight} = 0; 1 \text{ if weight} > 0\}$$

$$Arbiter = \{0 \text{ if not locked; } 1 \text{ if it is locked}\}$$

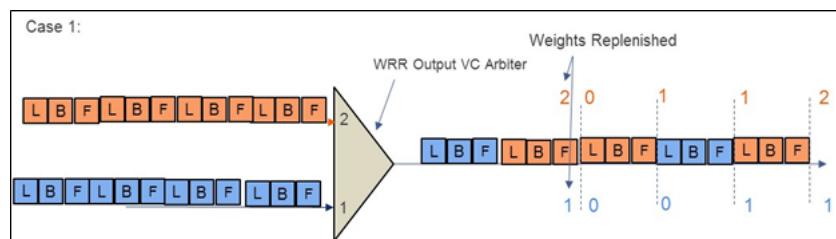
States	Input	Weight	Arbiter
S0	0	0	0
S1	0	0	1
S2	0	1	0
S3	0	1	1
S4	1	0	0
S5	1	0	1
S6	1	1	0
S7	1	1	1

No replenishment if the arbiter is locked and if any of the input is in state S6. Decrement the weight once the packet is serviced.

#### **11.3.4.2 Examples:**

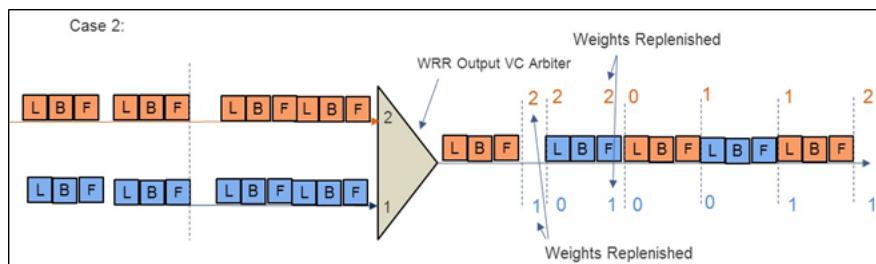
Case 1: Note how the weights are getting replenished once the weights go to zero. Also notice the packet interleaving between the input ports.

**Figure 11-1 WRR behavior.**



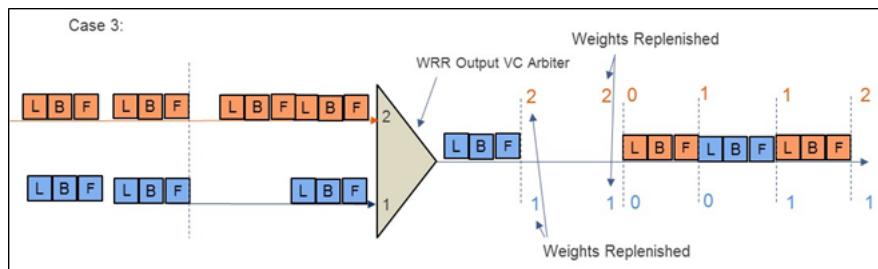
Case 2: Note how the weights get replenished when there are no valid PHITs to transmit on the links.

**Figure 11-2 WRR behavior on how weights get replenished when there are no valid PHITs to transmit on the link.**



Case 3: Note how the weights get replenished and the round robin behavior. Note that port 1 gets serviced, instead of port 0 after the weights are replenished.

**Figure 11-3 WRR behavior on how weights get replenished and RR behavior.**



### **11.3.5 Modified Work Conserving Deficit Round Robin (MWC\_DRR):**

## Behavior:

While WRR is simple to implement, it is not accurate in allocating bandwidth when the packet sizes are not uniform. With WRR, a Flow with large packet sizes will get more bandwidth than a flow with smaller packet sizes, even though both flows were allocated the same weight.

In order to account for the differences in packet sizes, we have developed the Modified Deficit Round Robin algorithm. The modified DRR algorithm is work conserving, that is, if there is a valid PHIT available it will be transmitted.

Each input Port is allocated a weight. The weight value represents the number of PHITs worth of bandwidth a port is allocated. Every time a PHIT from a port is serviced, the associated weight is decremented. Since packet sizes can vary and weights are fixed at system set up time, it may so happen that the packet being serviced has more PHITs than the remaining weight for that port. In this case the weight is allowed to go negative.

The weights are adjusted under two conditions:

- ▶ If none of the Input Ports is backlogged or has a valid PHIT to send.
  - ▶ Else if all the current weights are  $\leq$  zero and at least one port has a valid PHIT.
  - ▶ None of the port are able to make forward progress.

## Eligibility:

- ▶ The input Port/VC is eligible if:
    - ▶ It has a valid PHIT to send and the weight associated with the port is  $> 0$ .
    - ▶ If the weight associated with port  $\leq 0$ , and the current PHIT is not the First PHIT of the packet.

### Algorithm:

- Let the number of ports =  $N$
  - Let  $w_i$  be the weight assigned to  $inPort_i$
  - Let  $W = \sum_{n=1}^N (w_i)$
  - Initialization:  $u_i = w_i$

```

o   for( $i = 0 ; i < N; i ++$ ){
    §   if( $(inPort_i == eligible)$ ) /* Eligibility criteria above. Use  $u_i$ */
        §   {
            §   Service the port till the last PHIT
            §    $u_i = u_i - numOfPHITs serviced.$  /* Note:  $u_i$  can go negative */
        §   }
    o   }
    o   /* Weights replenish conditions */
    o   if( $no port has a valid PHIT$ )  $u_i = w_i$ 
    o   else continue {
        §   for( $i = 0; i < N ; i ++$ )  $u_i = \min(u_i + w_i, w_i)$ 
        §   break ((if at least one port becomes eligible) or (all  $u_i = w_i$ )
    }
}

```

### 11.3.5.1 State Table

Below is the state table for MDDR algorithm:

PHIT based:

$$input_i = \{0 \text{ if there is no valid PHIT}; 1 \text{ if there is a valid first PHIT}\}$$

$$weight_i = \{0 \text{ if weight} \leq 0 ; 1 \text{ if weight} > 0\}$$

States	Input	Weight
S0	0	0
S1	0	1
S2	1	0
S3	1	1

Replenish if none of the VCs/inputs is in State S3.

Packet based:

$$input_i = \{0 \text{ if there is no first PHIT}; 1 \text{ if there is a valid first PHIT}\}$$

$$weight_i = \{0 \text{ if } weight \leq 0 ; 1 \text{ if } weight > 0\}$$

$$Arbiter = \{0 \text{ if not locked}; 1 \text{ if it is locked}\}$$

States	Input	Weight	Arbiter
S0	0	0	0
S1	0	0	1
S2	0	1	0
S3	0	1	1
S4	1	0	0
S5	1	0	1
S6	1	1	0
S7	1	1	1

Do not replenish weights if the Arbiter is in locked state. Replenish when none of the inputs is in state S6.

### 11.3.6 Modified Non-Work Conserving Deficit Round Robin (NWC-MDDR):

The above MDRR algorithm is work conserving in nature. In an environment where the traffic can be bursty the system may get into a situation where the weights get replenished before all ports have used up their allocated bandwidth. In situations like this, opportunistic flows/ports may take more than their fair share. In order to mitigate this apparent unfairness, we have developed a Non-work conserving MDRR algorithm.

The difference between the WC and NWC DRR is in how the weights are replenished. For the NWC MDDR the weight replenishment should be done as follows (the rest of the algorithm remains the same).

/\* Weights replenishment \*/

*If (none of the ports are eligible){*

*continue to decrement the weights > 0 in a round robin fashion every cycle*

*If (all  $u_i \leq 0$ ){*

*for( $i = 0; i < N; i++$ ){*

$u_i = \min((u_i + w_i), w_i)$

*Break ((if at least one port becomes eligible) or (all  $u_i == w_i$ )*

*}*

*}*

*}*

#### 11.3.6.1 State Table

Below is the state table for NWMDR algorithm:

PHIT based:

$$input_i = \{0 \text{ if there is no valid PHIT; } 1 \text{ if there is a valid first PHIT}\}$$

$$weight_i = \{0 \text{ if weight} \leq 0 ; 1 \text{ if weight} > 0\}$$

States	Input	Weight
S0	0	0
S1	0	1
S2	1	0
S3	1	1

Replenish if none of the VCs are in S3 or S1. Continue to decrement the weight in a RR manner if the input is in state S1.

Packet based:

$$input_i = \{0 \text{ if there is no first PHIT; } 1 \text{ if there is a valid first PHIT}\}$$

$$weight_i = \{0 \text{ if weight} \leq 0 ; 1 \text{ if weight} > 0\}$$

$$Arbiter = \{0 \text{ if not locked; } 1 \text{ if it is locked}\}$$

States	Input	Weight	Arbiter
S0	0	0	0
S1	0	0	1
S2	0	1	0
S3	0	1	1
S4	1	0	0
S5	1	0	1
S6	1	1	0
S7	1	1	1

Do not replenish weights if the Arbiter is in locked state. Replenish when none of the inputs is in state S6 and S2.

### 11.3.7 None-Work Conserving Weighted Round Robin (NW-WRR):

None-Work Conserving WRR, is similar to Weighted Round Robin, except that

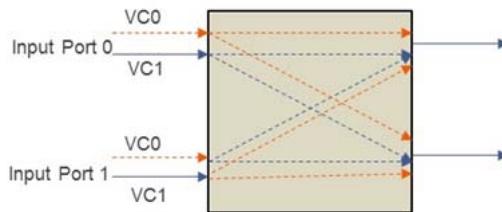
### 11.3.8 Weighted Fair Queueing:

TBD

## 11.4 Hierarchical Scheduling Algorithm:

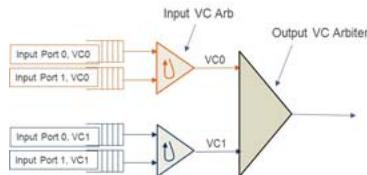
Hierarchical scheduling in Symphony is a two level hierarchy. A typical example of Hierarchical scheduling in Symphony is the scheduling done by VC-aware and VC based switches.

Let us first consider the scheduling done in the VC-aware and VC based switches. Figure 8.4, below shows a 2x2 switch having 2 VCs each.



**Figure 11-4 2x2 VC switch**

The output port arbitration is hierarchical as shown in F. The output port arbiter is hierarchical. The level 1 scheduler is an Input VC Arbiter. It selects the input port to be serviced next. The level 2 scheduler arbitrates between the eligible VC and picks VC that should be serviced next.

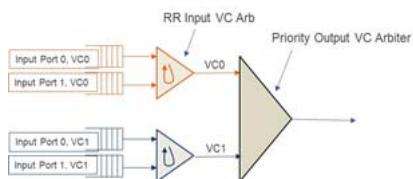


**Figure 11-5 Hierarchical Output Port Arbiter**

Note that level 1 input VC arbiter operates on packet basis. That is, once a VC is selected, it will remain selected for the duration of the whole packet.

### 11.4.1 Priority Round Robin

The priority round robin scheduler is shown in F, below.



**Figure 11-6 Priority RR Scheduler**

Eligibility:

A VC is eligible for arbitration at the output port arbiter, if at least one input port competing for the VC that has a valid PHIT.

**Behavior:**

The algorithm hierarchically selects the eligible input port per VC and then select the appropriate VC to be serviced. It should be noted that the second level scheduler arbitrates every cycle. This is how PHIT interleaving between VCs is achieved. Once an input port is selected by Level 1 scheduler, it locks this selection until the “Last” PHIT for the packet.

The algorithm does the following:

- First determines if the VC is eligible for arbitration.
- Picks the VC which wins the arbitration.
- Once the VC is chosen, the inputs ports competing for the VC are arbitrated on.
- The winning input port is then allowed to transmit.

For the ATU, the algorithm operates in a similar manner, except that the input ports are replaced with the queues in the CTL layer.

**Algorithm:**

- *Let  $p_i$  be the priority assigned to  $VC_j$*
- *Let  $p_i > p_{i+1}$*
- *Let  $N$  be the number of VCs on a given output port.*
- *Let  $M$  be the number of input ports mapped to the VC.*
- *Let  $vcList[N]$  be the list of VCs for a given output port*
- *When a VC becomes eligible do:*
  - *Add the VC to  $vcList[N]$*
  - *Sort  $vcList[N]$  in decreasing order of priority*
- *do( every cycle){*
  - *for( $i = 0; i < N; i + +$ ) {*
    - *if( $vcList[i] \rightarrow eligible$ ) {*
      - a. If an input port has already locked the VC.
        - i. Continue to service the input port.
      - b. Else

```

    i. Select the next eligible input port to service (round
       robin algorithm)

    c. Mark the input port as serviced

    }

    }

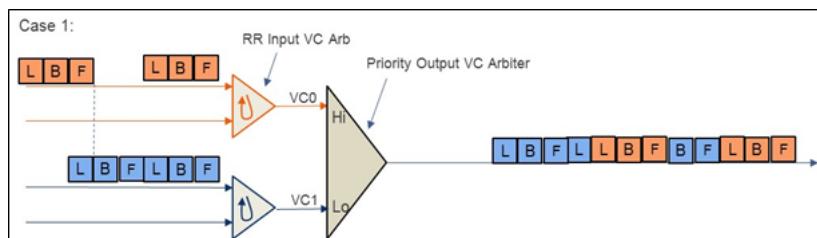
}

}

```

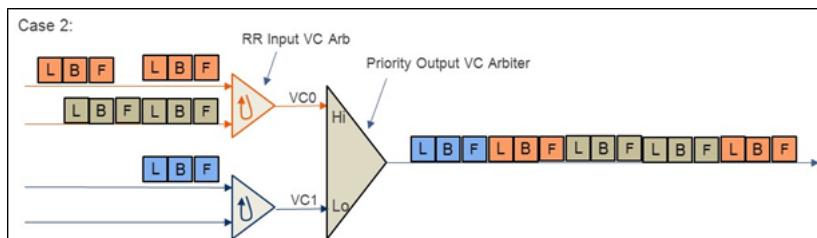
### 11.4.1.1 Examples:

Case 1: Figure 8-7, below describes the behavior of the PRR scheduler. Note the PHIT interleaving.



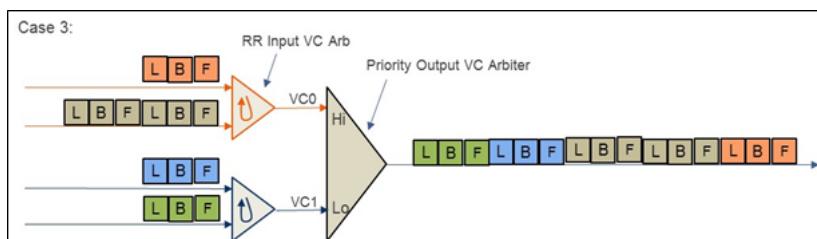
**Figure 11-7 Case 1: Priority RR scheduler behavior**

Case 2: Figure 5, below describes another example of the PRR scheduler. Note the RR behavior for high priority VC. No PHIT interleaving is allowed on the VC.



**Figure 11-8 Case 2: PRR scheduler behavior**

Case 3: Note the RR behavior of the lower priority VC. Again no PHIT interleaving is allowed on the VC.



**Figure 11-9 Case 3: PRR scheduler**

## 11.4.2 Hierarchical Round Robin:

Behavior:

In the switch, the algorithm selects from eligible VCs in a round robin manner. Once the VC is selected, the input ports competing for the VC are arbitrated on in a round robin fashion.

In the ATU, the algorithm operates in the same manner, except that instead of the input ports, the ATU has the queues in the CT Layer.

Eligibility:

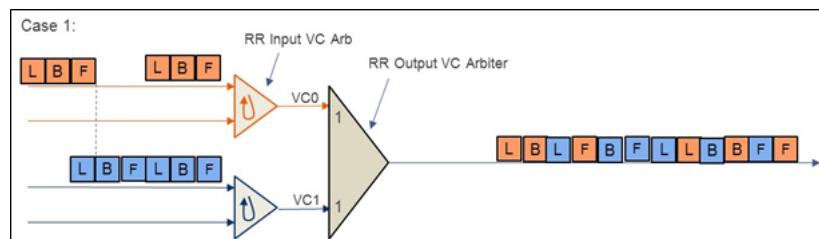
The output port VC is eligible for arbitration if it has a valid PHIT available on any input port mapped to the VC.

Algorithm:

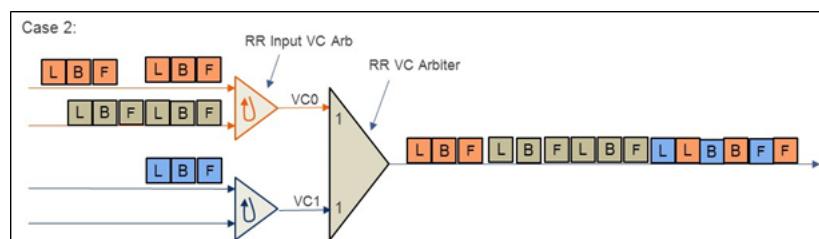
- . Let  $N$  be the number of VCs on a given output port.
- . Let  $M$  be the number of input ports mapped to the VC.
- . Let  $vcList[N]$  be the list of VCs for a given output port
- . Let  $input_i[M]$  be the list of input ports mapped to  $VC_i$
- . When a VC becomes eligible do:
  - o Add the VC to end of  $vcList[N]$  Do (every cycle)
  - o {
    - o Pick the next eligible output VC “i” to service from  $vcList[N]$ .
      - § If an input port has locked the VC, service the input port.
      - § Else service the next eligible input port from  $[M]$ , using the RR algorithm
        - Lock the input port for the VC.
    - § If PHIT == Last
      - Release input port lock on  $VC_i$ .

### 11.4.2.1 Examples:

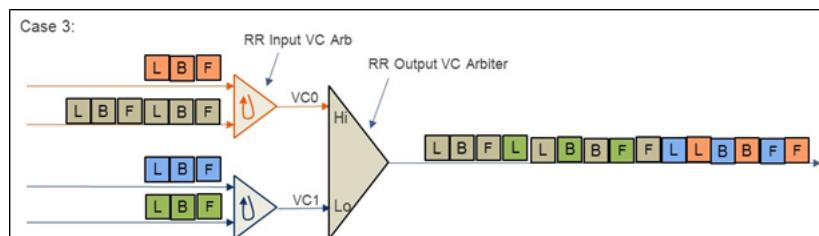
Case 1: Notice the PHIT level interleaving at the port.

**Figure 11-10 H-RR Algo behavior**

Case 2: Note packet level interleaving on VC0.

**Figure 11-11 H-RR behavior for Case 2**

Case 3:

**Figure 11-12 Case 3, H-RR behavior**

### 11.4.3 Hierarchical Weighted Round Robin:

**Eligibility:**

A VC is eligible for arbitration at the output port arbiter, if there is at least one input port competing for the VC has a valid PHIT.

**Behavior:**

The algorithm hierarchically selects the eligible input port per VC and then select the appropriate VC to be serviced. It should be noted that the second level scheduler arbitrates every cycle. This is how PHIT interleaving between VCs is achieved. Once an input port is selected by Level 1 scheduler, it locks this selection until the “Last” PHIT for the packet.

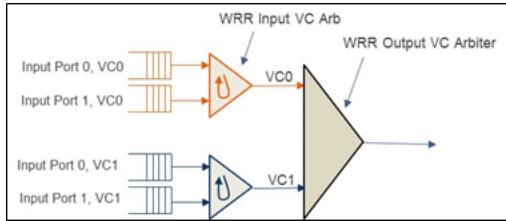
The algorithm does the following:

First determines if the VC is eligible for arbitration.

Picks the VC which wins the arbitration.

Once the VC is chosen, the inputs ports competing for the VC are arbitrated on.

The winning input port is then allowed to transmit. For the ATU, the algorithm operates in a similar manner, except that the input ports are replaced with the queues in the CTL layer.



**Figure 11-13 H-WRR scheduler**

Algorithm:

Let:

- .  $N = \text{number of } VCs$
- .  $M_i = \text{number of input ports mapped to } VC_i$
- .  $t_{ij} = u_{ij} = \text{weight assigned to inPort}_i \text{ mapped on to } VC_j$
- .  $y_i = w_i = \text{weight assigned to } VC_i$
- .  $U_{ij} = \sum_{i=1}^M u_i$   $W = \sum_{i=1}^N w_i$
- .  $\text{portServiceable}_i = \text{port } i \text{ is serviceable iff it has a valid PHIT \&& } u_i > 0$
- .  $\text{vcServiceable}_i = VC_i \text{ serviceable iff } VC_i \text{ has a serviceable input port}$
- .  $vcB_i = \text{bandwidth allocation of } VC_i: vcB_i = \frac{w_i}{W} \times (\text{Output port bandwidth})$
- .  $b_i = \text{bandwidth allocation of input port } i = \frac{u_i}{U} \times (vcB_i)$
- .  $\text{lastServiced} = \text{last input port serviced by the scheduler}$
- .  $\text{vcLastServiced} = \text{last } VC \text{ serviced by the scheduler}$
  
- . For every cycle do:
  - o While( $\text{vcServiceable}_i == \text{true}$   $\forall i$ ) {if ( $\text{vcServiceable}_{\text{vcLastServiced}}$ )
  - {

```

for( i = lastServiced; i < N + lastServiced; i ++){

    if(input port i locked && no valid PHIT){

        Break ; } /*break out of the loop and do not do
        anything */

    else if (portServiceablei)

        { if( PHIT == LAST){

            service PHIT;

            ui = ui - 1;

            lastServiced = (i + 1)modM }

        else service PHIT ;

    }

}

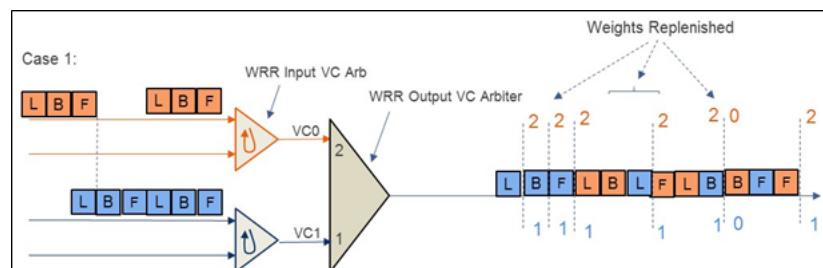
```

## Weight Replenishment:

If ( $\neg portServiceable_i \forall i$ ) {  $u_i = t_i$  }  
     if ( $\neg vcServiceable == true \forall i$ ); {  $w_i = y_i$  }

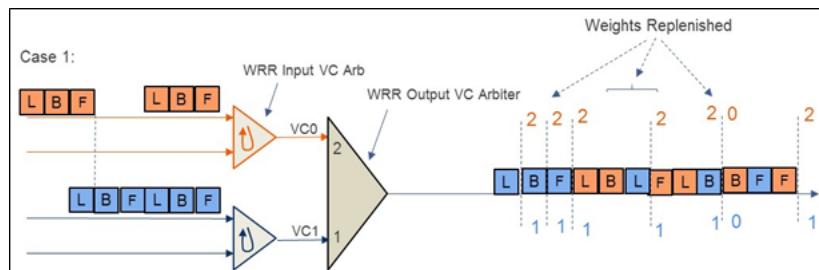
#### **11.4.3.1 Examples:**

Use Case 1: Note how the weights are replenished and the RR behavior between VC0 and VC1.



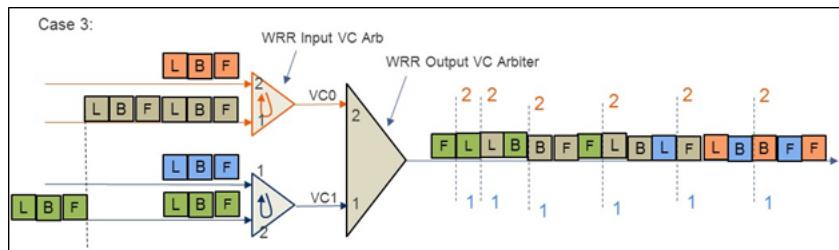
**Figure 11-14 Case 1: HRR behavior**

Use Case 2: Note how weights on the input VC arbitration update.



**Figure 11-15 Case 2: H-WRR behavior**

Use Case 3: Note how the different packets are serviced.



**Figure 11-16 Case 3: H-WRR behavior**

#### 11.4.4 Hierarchical Priority Weighted Round Robin:

Hierarchical PWRR is similar to Priority Round Robin, except that the first level scheduler is WRR and the second level is Priority.

With priority RR/WRR the VC arbiter will always pick the highest priority VC, there is a possibility that the high priority traffic may starve out low priority traffic, even if the high priority traffic is rate limited. A typical case could be that the high priority flow target stalls for several cycles. During this stall, the arbiter will always pick the high priority PHIT even though the PHIT will not make any progress. To avoid starvation of low priority traffic, there is a starvation counter in the Priority arbiter. This starvation counter will service the low priority once the count goes past a configured number.



# 12

## Traffic Classes

Traffic within an SoC is segmented into six distinct categories. These categories may or may not simultaneously exist in the SoC.

These categories are:

- ▶ Network IO Traffic -- **Bursty with bandwidth requirements and latency limits.**
  - ◆ Ethernet traffic needs to be buffered as soon as it enters the SoC. Otherwise buffer overflow can happen on the connected Ethernet link.
- ▶ Traffic generated by the application running on the host processor – **Non-critical Bursty but low latency.**
  - ◆ Application stalls due to latency in the processor subsystem lowers the overall system performance
- ▶ Traffic generated by mission critical components – **Strictly real-time.**
  - ◆ Airbag systems, auto pilot systems in cars, pace makers
- ▶ Traffic generated by devices that is **Isochronous** (periodic).
  - ◆ Video display traffic.
- ▶ Low speed Peripheral IO – **Low bandwidth and looser latency limits.**
  - ◆ I<sup>2</sup>C, UART, ..
- ▶ Traffic generated as a result of interacting with Hardware accelerators – **Bandwidth Guarantees.**
  - ◆ Security engine.
  - ◆ Compression/decompression engine.



# 13

# Quality of Service

This chapter will discuss the different mechanism Symphony has, which can be used in the Interconnect to maintain and meet Quality of Service requirements of different flows in the Interconnect.

## 13.1 QoS System Overview

As was described in the Section on Traffic Classes, there can exist in the SoC, different classes of service with differing Quality of Service requirements. These different may exist at the same time in the SoC.

The system designer is faced with two diametrically opposing requirements at the system level. These requirements are: 1) Meet the QoS requirements for the different types of flows in the Interconnect; and 2) reduce the overall Interconnect area and power.

Guaranteeing end-to-end QoS and meeting the above stated requirements, depends on the following:

- ▶ Choosing the appropriate Interconnect Topology.
- ▶ Appropriately mapping and routing the flows through the Interconnect.
- ▶ Employing appropriate transient congestion controls
- ▶ Employing appropriate sustained congestion controls.

In what follows, contains a general discussion on each of the above and how Symphony implements them.

## 13.2 Topology

Interconnect topology refers to the relative placement and arrangement of different elements such as ATUs, Switches, Adapters etc, in the SoC.

Interconnect topology could be regular as in Mesh, Torus, Butterfly or irregular depending on which type of topology provides the optimum area, power and QoS values.

The topology of the Interconnect directly impacts the throughput and delay of traffic flowing through it as well as the cost. For example, high radix switches (switches with large number of ports) in the network reduce the average number of hops a packet has to take to get to the destination thereby reducing the delay, primarily because less number of switches have to be traversed. But may not be suitable because of their size and placement issues. On the other hand smaller radix switches add to the delay and reduced network throughput but may be better from a placement point of view.

Symphony will provide options for the customers to determine the type of solution they are looking for. Details are TBD.

## 13.3 Flow Mapping

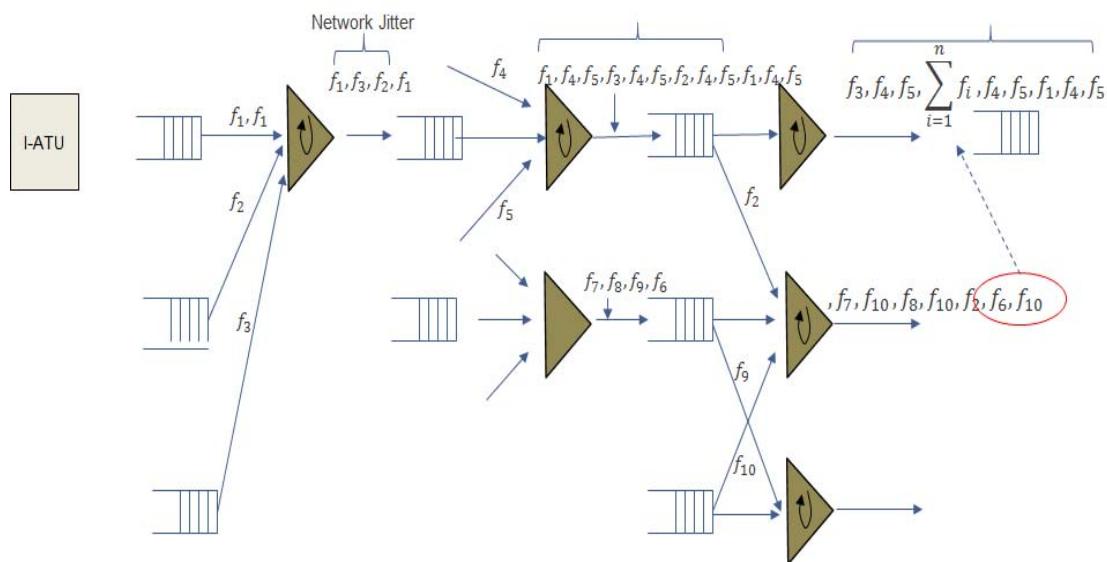
Flow mapping refers to the process of determining the appropriate path for a particular flow such that the flow meets its QoS requirements. We will refer to a flow as “schedulable” if there exists a mapping that allows the flow to meet its QoS requirements.

Flows in the Interconnect can be separated into primary interfering flows and secondary interfering flows. Consider the network shown in the Figure below. If the flow of interest is  $f_1$ , then all the flows that share at least a link (or VC) with  $f_1$  would be referred to as primary interfering flows. In the Figure below, flows  $f_2, f_3, f_4$ , and  $f_5$  are primary interfering flows. Flows that do not share a link or VC with  $f_1$  but share a link or VC with any of the primary interfering flows are referred to as the secondary interfering flows. In the Figure below,  $f_6, f_7, f_8, f_9$ , and  $f_{10}$  are all secondary interfering flows.

Notice the increase in delay jitter that packets from  $f_1$  experience as they flow through the network. Also notice that on the last switch the jitter dramatically increases as indicated by the summation of jitter introduced by all the secondary flows. The impact of secondary flows is because of  $f_2$  as it interacts with other flows in another switch.

The impact of secondary flows is more pronounced in a network of Flow-Through switches.

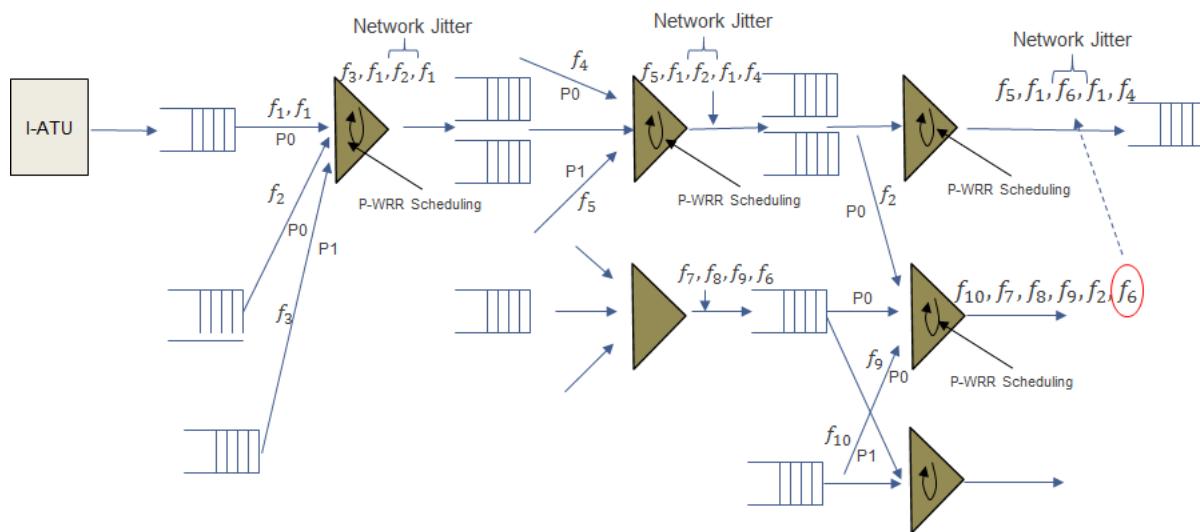
**Figure 13-1 Primary and Secondary Interfering Flows**



There are ways to reduce the jitter especially for latency sensitive flows. One straight forward approach would be to not map as many flows on the links that  $f_1$  is mapped to, but that would reduce the link utilization and consequently increase the overall cost. Other ways are to use, which may not reduce the overall link utilization, are to use the appropriate type of arbitration algorithm in the switches and ATU as well as use Virtual Channel capability. The Figure below illustrates that by using Virtual Channels and Priority Weighted Round Robin the network jitter is significantly reduced while maintaining the same level of link utilization.

Flow mapping process will determine which flows are schedulable on what paths.

**Figure 13-2 Network jitter reduction by using appropriate mechanisms in the network**



## 13.4 Routing:

See Routing Section for details.

## 13.5 Transient Congestion Controls

Transient congestion control mechanism deal with short term congestion. In interconnects, transient congestion is caused when two or more packets contend for the same output port. Symphony deals with this transient congestion by using the appropriate scheduling algorithm in the arbiters present in the ATUs, and the switches, and link-by-link flow control.

In Chapter on Scheduling, we discuss the various scheduling algorithms that would be part of the repertoire of scheduling algorithms available to the arbiter in Symphony. These include simple Round Robin, Weighted Round Robin, Priority Weighted Round Robin as well as hierarchical scheduling algorithms.

These scheduling algorithms will enable us to partition bandwidth between different flows, input ports and/or Virtual Channels.

In the Flow-Through switch network with Virtual Channels, our scheme allows PHIT/FLIT interleaving, that is, PHITs from different virtual channels can interleave. The interleaving of PHITs/FLITs reduces the latency and network jitter.

Symphony scheduling algorithms are also designed in a way to prevent deadlocks, reduce jitter and apportion bandwidth in weighted manner. Priority scheduling has the potential to cause starvation. However, this is prevented by implementing end-to-end credit management and starvation counters wherever Priority scheduling is implemented.

Symphony has link by link flow control all the way from the I-ATU to the T-ATU.

## 13.6 Sustained Congestion Controls

Sustained congestion is defined as congestion that persists for a much longer period of time than transient congestion.

Symphony has two sustained congestion control mechanisms. These are: 1) Rate controller in the I-ATUs; and 2) End-to-end credit management.

The Rate controller as defined in the Credit Management section, will limit the amount of traffic an I-ATU can inject in to the Interconnect. The rate controller supports different of traffic profiles which can be configured at initialization time. The rate controller although restricts the injection rate, is a static control. That is, once set it cannot be modified (at this moment, see the Enhancement section in the Credit Management Section).

Credit Management on the other hand provides dynamic control of the traffic injection rate in to the Interconnect. The credit management mechanism will dynamically adjust the injection rate based on the congestion in the interconnect and the slave by regulating the return of the credits to the Initiator.

Both controls are required to prevent sustained congestion in the network.

## 13.7 Virtual Channels (VC)

Virtual channels are explained in detail in the Switch section. In this section we will briefly explain the usage of VCs.

Virtual Channels offer the following:

- ▶ Traffic isolation. That is, different traffic classes can be mapped to different VCs and by using appropriate scheduling algorithms in the ATU and Switch arbiters the Interconnect can guarantee bandwidth and delay.
- ▶ Increase Interconnect throughput by reducing HoL. That is, by adding VCs we reduce Head-of-line blocking and allow forward progress on the VCs that are not blocked.
- ▶ Deadlock avoidance. VCs can be used to prevent deadlocks if a single network is used for both request and response messages.
- ▶ VCs enable virtual networks to be implemented as well.

Symphony will support multiple flavors of VC implementations. In R1 release, Symphony will offer Flow-Through VC aware switches. These switches are VC aware, but do not have any buffering to hold PHITs/FLITs. Later releases of Symphony may support buffered VC switches if needed.

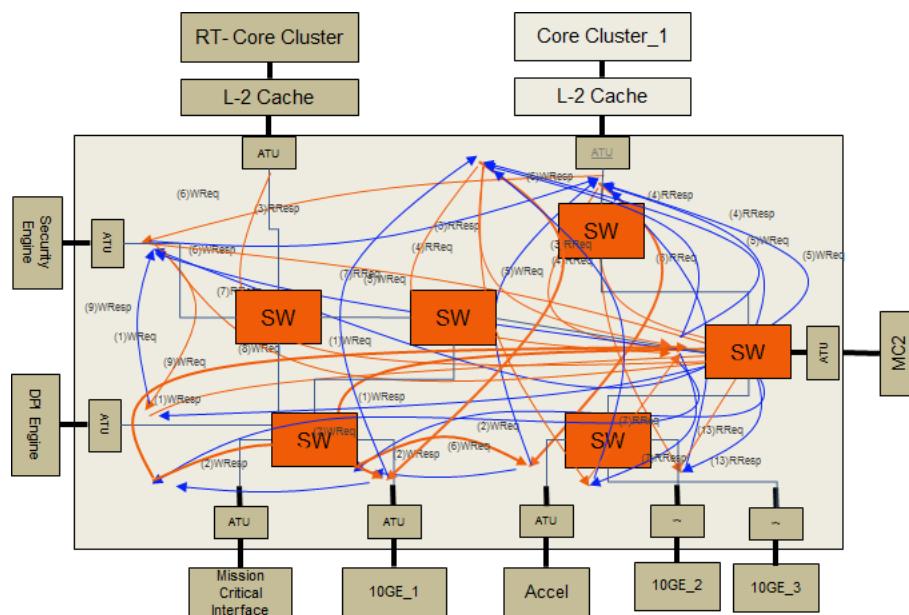
## 13.8 Flow Mapping Revisited

Consider the SoC shown in the Figure below. The example SoC consists of Real-time flows, low latency flows, guaranteed bandwidth flows and best effort flows. Real-time flows are from the real-time core cluster, mission critical IO, and GE\_3 to memory are real-time flows. Low latency flows are primarily from the Core Cluster\_1. Similarly guaranteed bandwidth flows are from the accelerators to memory.

Below is a flow mapping table example that will be provided by Symphony.

**Table 13-1**

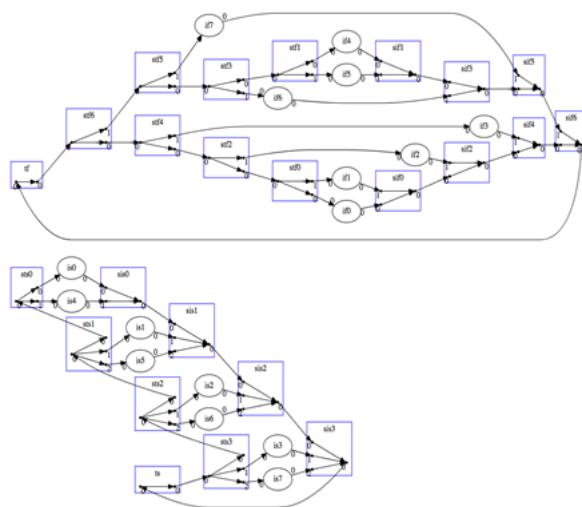
**Figure 13-3 Example of an SoC and the different flows in the SoC**



## 13.9 Examples

### 13.9.1 Transient Congestion Controls:

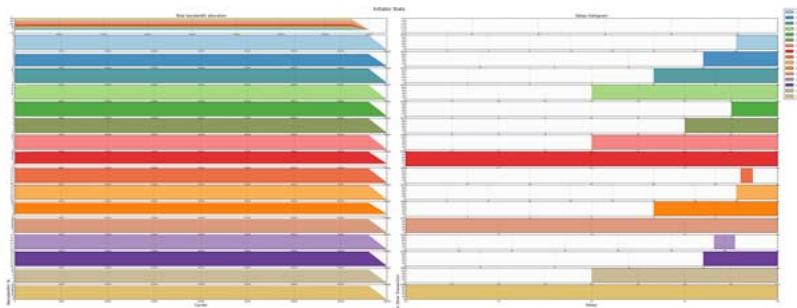
Let us start with the example as shown in the Figure below:



**Figure 13-4 Daisy Chain Network - Two separate Interconnect vs. Single Interconnect**

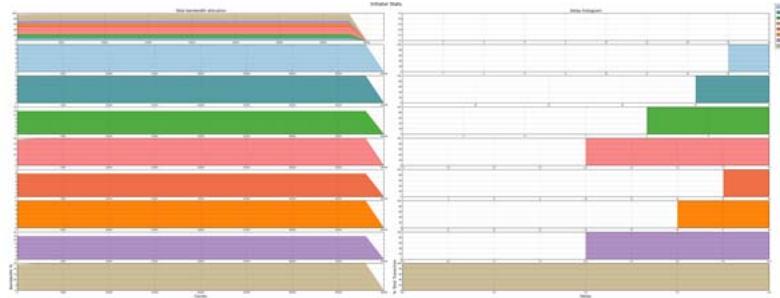
The diagram on the top, represents two separate daisy chain network while the one in bottom represents a single daisy chain network with the same number of Initiators and Targets. The Initiators are equally divided between High Priority ISO Initiators (0-3) and Low Priority Best Effort Initiators (4-7)

The Figure below shows the throughput and latency experienced by the different Initiators in the Interconnect. The top eight results are from the two separate networks and the bottom eight results are for the single network. Notice the difference in performance in having two separate networks as opposed to a single network.

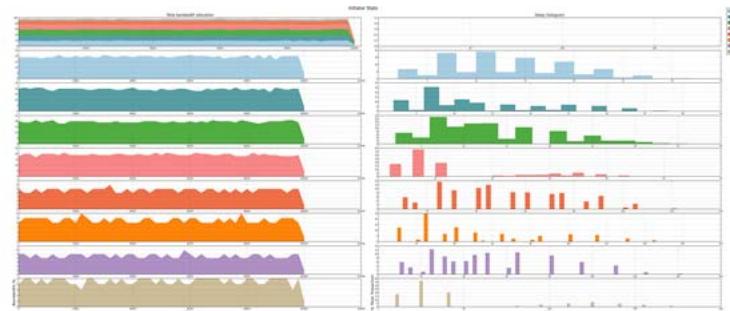


**Figure 13-5 Raw Performance Results -- Top 8 graphs are for two separate networks. Bottom 8 are for single network**

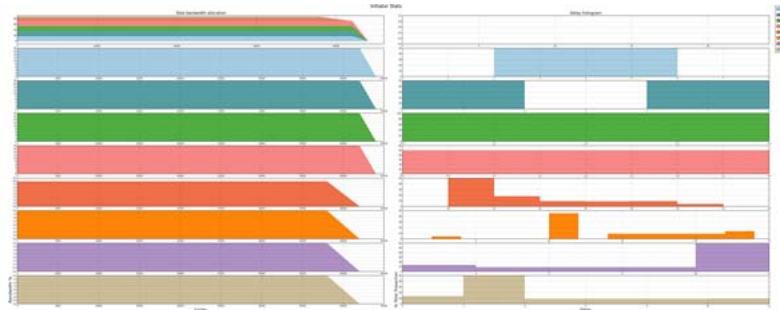
Next we add VCs to the single network and evaluate the performance of the network. In the Figure below we present these results. Notice that the performance of the single network with VCs is similar to that of two separate networks as shown above. Thus illustrating that VCs in fact enable isolation of traffic and performance.

**Figure 13-6 Adding VCs to the single network**

Next we able WRR scheduling in the arbiters instead of the simple Round Robin algorithm (on the single network). Our objective is to be able to allocate 80% of the bandwidth of the Interconnect to High Priority Initiators, split evenly between them and the rest 20% between the Low priority Initiators evenly. Using hierarchical WRR in the switches we are able to allocate 80% of the bandwidth evenly between High Priority initiators and the remaining 20% between the Low Priority Initiators. This result is shown in the Figure below.

**Figure 13-7 Adding VC + Hierarchical WRR to the single network**

Notice that while, we were able to allocate the bandwidth as desired, we did not reduce the latency for high priority initiators. In the Figure below we show the results when we enable Priority WRR scheduling in our arbiters and rate limit the high priority initiators.

**Figure 13-8 Adding Priority WRR instead WRR**

## 13.9.2 Sustained Congestion Controls

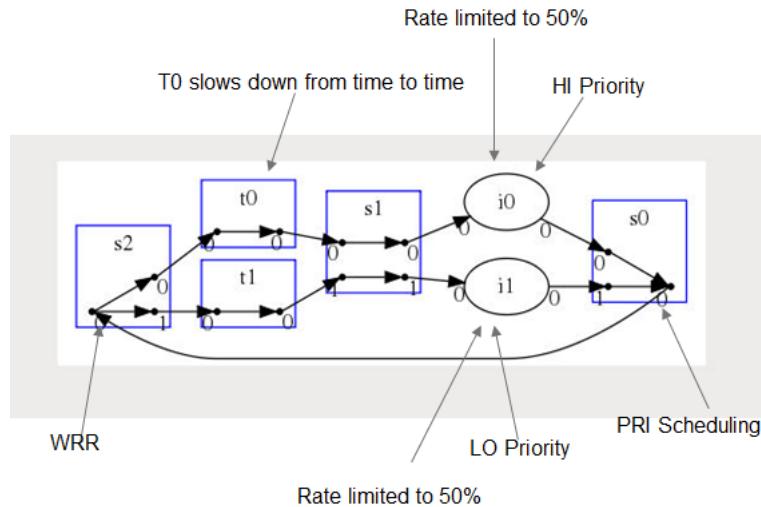
Symphony has the following sustained congestion controls:

- ▶ Rate limit
- ▶ Credit Management

In Chapter x, on Credit Management we explain credit management scheme and rate limiter in detail. In this section, performance results will be presented.

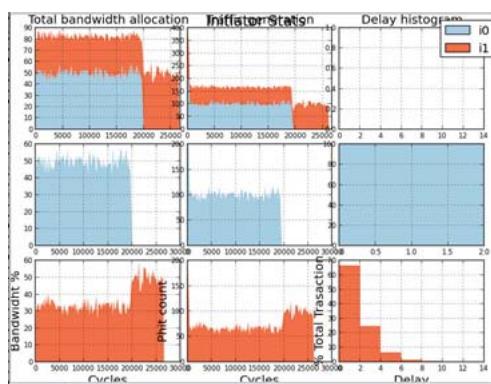
We start with a simple network as shown in the Figure below:

**Figure 13-9 Simple Network to demonstrate Credit Management**

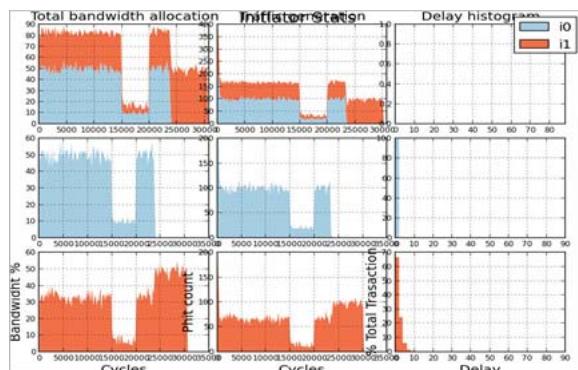


In the network above, I0 and I1 are high and low priority initiators. Switch S0 implements priority scheduling, S2 implements WRR scheduling. I0 only sends traffic to T0 and I1 only sends traffic to T1. The injection rate from both Initiators is capped at 50% per initiator using our rate limiting mechanism. If we were to allow I0 to send at 100% injection rate, then I1 would not be able to send any traffic. Figure below shows the performance of the network and the throughput of each Initiator.

**Figure 13-10 Performance of I0 and I1**

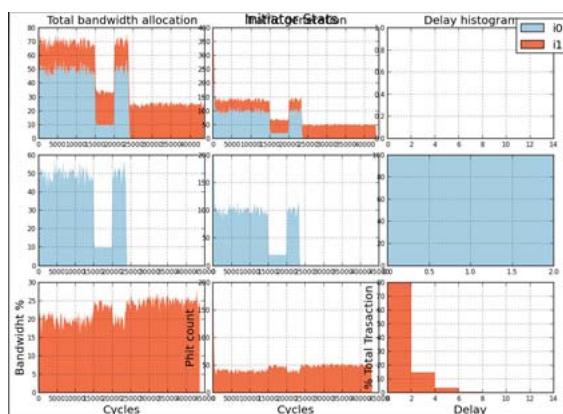


Next we would like evaluate Interconnect performance if for some reason target T0 slows down and instead of accepting packets every cycle it accepts every 10 cycles. Figure below illustrates the impact of T0 slowing down.

**Figure 13-11 Performance of the network when T0 slows down**

Notice that while throughput of I0 reduced to 10% because the T0 slowed down. However, this reduction also impacts the throughput of I1. This is scenario that can happen in the interconnect and will impact the performance of the flows sharing the same link or Virtual Channel.

To remedy the above situation, we implement credit management. Both Initiators are allocated credits. The Figure below illustrates the impact of using credit management on the performance of the system.

**Figure 13-12 Performance of the Interconnect when Credit Management is enabled**

Notice that I1 does not lose any bandwidth when credit management is enabled on both. Note, the number of credits allocated to I1 is adjusted so that it will not exceed 20-25%.

## 13.10 Topology Description and Flow Mapping

| See ymphony Topology Chapter

## 13.11 Summary

Symphony provides a wide set of tools to manage quality of service in the Interconnect. The breath of tools and mechanisms available is sufficient and complete.

| As discussed in the above section, different mechanisms would be used in different situations to maintain QoS across the Interconnect for different flows.

## 13.12 Performance Measurements:

The following performance measurements should be made while reporting performance results for Symphony:

1. Latency measurement
  - a. Min, Max, Mean, Histogram
  - b. Latency measurements per class of service
  - c. Latency measurements per transaction type
2. Throughput measurement
  - a. Min, Max, Mean , Histogram
  - b. Throughput per transaction type (e.g.: Read/Write/etc.)
  - c. Throughput per class of service
3. Link utilization
  - a. Idle
  - b. Busy
  - c. Bubbles

### 13.12.1 Latency Measurement:

Symphony needs to enable the following types of latency measurements in the system:

- a. Initiator to Target Latency calculation
- b. Target to Initiator Latency calculation
- c. Round trip latency calculation
- d. Latency measurements per class of service

Unidirectional Latency is defined as the time that transaction takes traversing the network (single or multiple Fabrics) from the Source node to the destination node. Latency can be calculated from the time the Protocol Socket puts the first beat of the transaction on the input link to the Initiator ATU to the time that Target puts the transaction on the outgoing link to the Socket. Latency could also be defined at the packet level, that is, the time the first beat of the packet is put on the ATP interface by the initiator to the last beat of the packet received at the Target. Below we provide a formal definition of calculating the latency of transactions and packets.

Latency calculation is defined as follows:

- ▶ End-to-end Socket Latency = (Time the last beat of the transaction put on the Socket interface by the Target ATU) minus (Time the first beat of the transaction is received by the Initiator ATU from the Socket). The calculations should be done in both directions, i.e., from the Initiator to the Target and from the Target to the Initiator.
- ▶ End-to-end Transport Latency = (Time the last beat of the packet is received by the Target ATU at the ATP interface) minus (Time the first beat of the packet is put on the ATP interface by the Initiator ATU). The calculations should be done in both directions.
- ▶ First-to-First Socket Latency = (Time the first beat of the Transaction is put on the socket interface by the Target ATU) minus (Time the first beat of the transaction is received by the Initiator ATU from the socket)

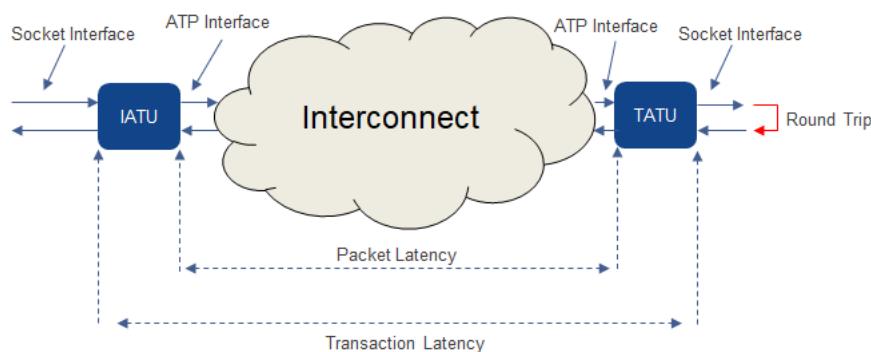
- ▶ First-to-First Socket Latency = (Time the first beat of the Packet is received by the Target ATU on the ATP interface) minus (Time the first beat of the packet is put by the Initiator ATU on the ATP interface)

Round Trip latency is calculated as follows:

Round Trip Socket Latency= (Time the request command is received by the I-ATU) minus (Time the last data beat/Response is put on the Socket interface by the Initiator)

Round Trip Transport Latency=

Latency calculation is further segmented into Write and Read Socket latency.



**Figure 13-13 Latency calculation**

The latency should be measured under the following conditions:

1. Zero Load Latency: Under this condition, the flow under observation is the only flow in the network. The purpose of this measurement is to determine the best case latency in the system.
2. Saturation Load Latency: In this test the network is loaded in such a way that from every initiator there is always a transaction ready to be sent. The latency of transactions/packets for the flow in question are calculated. This scenario represents the worst case latency through the system.
  - a. If multiple classes of traffic are supported in the latency of each Classes of Service should be observed..
  - b. Measurements should be taken at multiple points as the load is gradually increased to the point when the throughput of the system saturates and does not increase any further as the load is increased.
3. Operating Load Latency: In this test the network is loaded at the defined “operational loading” level. This loading level is just below the value where the Transaction latency at the socket level starts to rise exponentially. This test is customer specific. The performance model should enable it, but as a general rule this result will not be provided by Arteris. Partly covered in (2).

## 13.12.2 Throughput Measurement:

Throughput measurements are done at both the transaction and packet level. At the transaction level, for ABMA AXI protocol, the throughput calculations are done with respect to the type of transaction, such as

Read transaction, Write transaction, Atomic etc. Other protocols can have their protocol specific throughput calculations done.

Throughput at the transaction level implicitly includes the round trip throughput. That is, write throughput implicitly includes the write response throughput and read throughput implicitly includes the read request throughput. This is due to the restriction on the number of outstanding transactions in the system. The Initiator will not send any more read or write transactions to the network if the number of outstanding transactions is greater than the limit set by the system.

For now we limit the definitions below to read and write transactions:

**Write Bandwidth** = Number of Bytes sent by the initiator into the network/Number of Cycles (count starts with first data beat at the initiator till the last data beat at the initiator)

**Read Bandwidth** = Number of Bytes received by the Initiator from the network/Number of cycles

Socket level and packet level throughput could be measured at the Socket interface and the ATP interface respectively.

Measurements should be able to be done for a particular window size where the observation window size should be configurable. For example, video traffic can be composed of a peak period and an idle period. This type of traffic will be captured by the On-Off traffic model and with measurements made over the on-off period of the traffic.

Throughput should be measured under the following conditions:

1. Zero Load Throughput: This test will give the maximum throughput that the flow in question can achieve. The only flow in the system is the flow in question.
2. Saturation load throughput: This measurement is carried out when all the initiators are greedy and sending as much traffic as they can.
3. Operating load throughput: This is the throughput, beyond which the latency of the system increases exponentially. As with the latency measurement, the performance model should be instrumented to support this measurement by the customer.
4. Traffic generated should consist of different Burst Sizes:
  - ◆ Single beat
  - ◆ Cache size
  - ◆ Mid size burst size
  - ◆ Mix of Writes and Reads transactions so as to evaluate cross traffic impact.
  - ◆ Back to back transactions to measure max achievable bandwidth.

### **13.12.3 Product Specific Measurements**

Product specific measurements such as buffer fills in FIFOs, Cache miss rates, impact of the number of outstanding transactions on the performance of the system etc. will be covered in detail in a separate document.

### **13.12.4 Sampling**

Data sampling is important to get statistically reliable data. Symphony will collect throughput data in sample sizes of 1000-10,000 cycles. There would be several of these samples collected. Measurement would include capturing the histogram, so that we can get the following values: Mean, Min, and Max.

The throughput obtained during smaller cycle numbers would help in fine tuning buffer depths. One would expect the variance in measured data to be high for smaller cycles than for larger number of cycles.

A confidence interval of 95% is generally used as the standard in reporting results. The confidence interval would be calculated as follows:

1. Step 1: Subtract the confidence interval from 1 and divide by 2:  $(1-0.95)/2 = 0.025$
2. Step 2: Subtract the 0.025 from 1 and calculate the Z-score = 1.96
3. Step 3: Confidence Interval: Where Z is the Z-score value,  $\sigma$  is the standard deviation and  $n$  is the sample size.

$$95\% \text{ Confidence Interval} = \text{Mean} +/ - Z * \frac{\sigma}{\sqrt{n}}$$

## 13.12.5 Traffic Generation

Three different types of traffic generation are required along with the ability to load a trace file and generate traffic according to the events in the trace file.

The three types of traffic generation needed are:

1. On-Off traffic model to represent bursty traffic scenarios where multiple transactions are sent by the Master back to back for the burst duration. The burst size of the transaction is also varied based on an average burst size. An example of bursty traffic is video traffic.
2. Greedy traffic model where the initiators have a transaction always ready to be transmitted if the system would allow them. The data size can be varied for AXI transactions. For cache coherent transactions the data size can be fixed.
3. Periodic rate limited traffic
4. Workload/Trace files used to generate traffic
5. Number of outstanding transactions

## 13.12.6 Network Setup

Network topology and setup has a big impact on the performance results. While the customer would be interested in obtaining performance results based on their target network and traffic pattern, it seems nevertheless important that when Arteris reports results they are gathered on a standardized network and test/simulation setup. As an example, to measure the saturation throughput in the system, initiators should generate traffic in a unified manner. Below is an example configuration and traffic profile setup:

Network Configuration: Regular Mesh network.

Initiator Traffic Profile: Uniformly chooses the destination from among the valid Targets.

**Initiator/s Under Observation:** Initiator/s send traffic to designated Targets. That is, a flow is created and its throughput and latency are observed.

**Traffic Loading:** Traffic is directed in a such as manner so as to create congestion on the link/s. Traffic is also directed to create cross-traffic impact on the flow under observation.

## 13.13 Flow-to-VC Mapping

The packet header has two fields associated with quality of service. These two fields are the priority field and the QoS field. The priority field defines the priority of the packet with respect to other packets. A flow can have packets that can have different priority values. The QoS field is dual purpose. That is, it reflects the QoS level the flow needs and the Virtual Channel the flow is mapped to on that link. QoS value of the packets in a flow does not change for the duration of the flow.

The initiator ATU will make sure that for AXI protocol, the AXI ID has precedence over the QoS field. That is, if a transaction comes with an AXI ID, which is the same as another outstanding transaction or one that is already queued in the ATU, then regardless the QoS value of the current transaction, it will be mapped to queue that was assigned to the previous transactions. The Initiator ATU has to be configured to map the incoming QoS value from the AXI interface into the QoS value that will be used internal to the network. This mapping will also determine the flow queue inside the ATU to which the transaction will be mapped to and the VC in the Packet layer that will carry the packet corresponding to this transaction.

Symphony switches can be configured to look at any field in the packet header to make the forwarding decision. The two relevant fields for making forwarding decision are the QoS field and the route field. The route field determines the forwarding port on the switch and the QoS field determines the VC the flow is mapped to. The master arbiter on each switch is then configured appropriately to schedule the VCs. The switch supports two level hierarchical scheduling. The first level arbitrates between the input ports mapped to the VC and the second level arbiter arbitrates between the different VCs on the link. Multiple scheduling algorithms are supported in Symphony. These are: Round Robin, Weighted Round Robin, Priority, and Priority Round Robin.

The path a flow will take in the network is determined by the routing algorithm. The routing algorithm also ensures that the flow path does not cause any deadlock in the system. The routing algorithm also makes sure that there is enough capacity in the link accommodate the flow.

Once the flow has been mapped by the routing algorithm on to the links in the network, the second step in mapping then is to map the flow to the appropriate Virtual Channel.

The number of virtual channels in the system depends on the following:

1. Number of QoS classes in the System.
2. Virtual channels needed for deadlock avoidance.
3. Separation of traffic based on user determined criteria

For mapping purposes, Virtual Channel usage for deadlock avoidance is orthogonal to QoS. Hence it will not be discussed here. It would suffice to say that the number of virtual channels used for QoS or user defined separation criteria, will be duplicated on the links where deadlock is removed by using VCs.

The number of VCs will be deduced from the overall number of priorities or classification of flows done by the user. The user defined flows need to be bucketized in terms of QoS or user defined criteria and hence VCs. Only a small set of VCs would make sense. We recommend a maximum of 4VCs in the system. The number of VCs varies based on how the flows are mapped. For example, if on a particular link only the highest priority flows are mapped then only one VC would be required.

As was described earlier in the Traffic Classes chapter, flows can be categorized as either being:

1. Latency sensitive. Latency sensitive flows require that they get the best latency through the network. They also require that the latency is not dependent on network load.

2. Latency bound. These are flows that have to maintain a certain latency between source and destination. Such are IP like the display driver, and other IO devices. The latency bound is to avoid buffer underflow. The other requirement is that the latency bound is observed even during times of congestion. However, the amount of traffic from latency bound sources is known. For example, the traffic going to the display driver can be easily estimated. Similarly, the traffic from IO devices such as Ethernet, etc. is known.
3. Bandwidth sensitive: These flows require the minimum guaranteed bandwidth. Examples of these flows would be flows feeding traffic to a video encoder for example.
4. Best Effort flows: These flows are best effort flows and will consume as much traffic as they can get. They are not latency sensitive or bandwidth sensitive.

Given the above flow classification, the choices are to use 2, 3 or 4 VCs. Note that VCs come at a cost. Adding VCs to the network increases the area and power of the Fabric. Since the high priority flows can starve out lower priority flows, it is recommended that they be rate controlled.

**Table 13-2 VC mapping table for 4-Traffic classes**

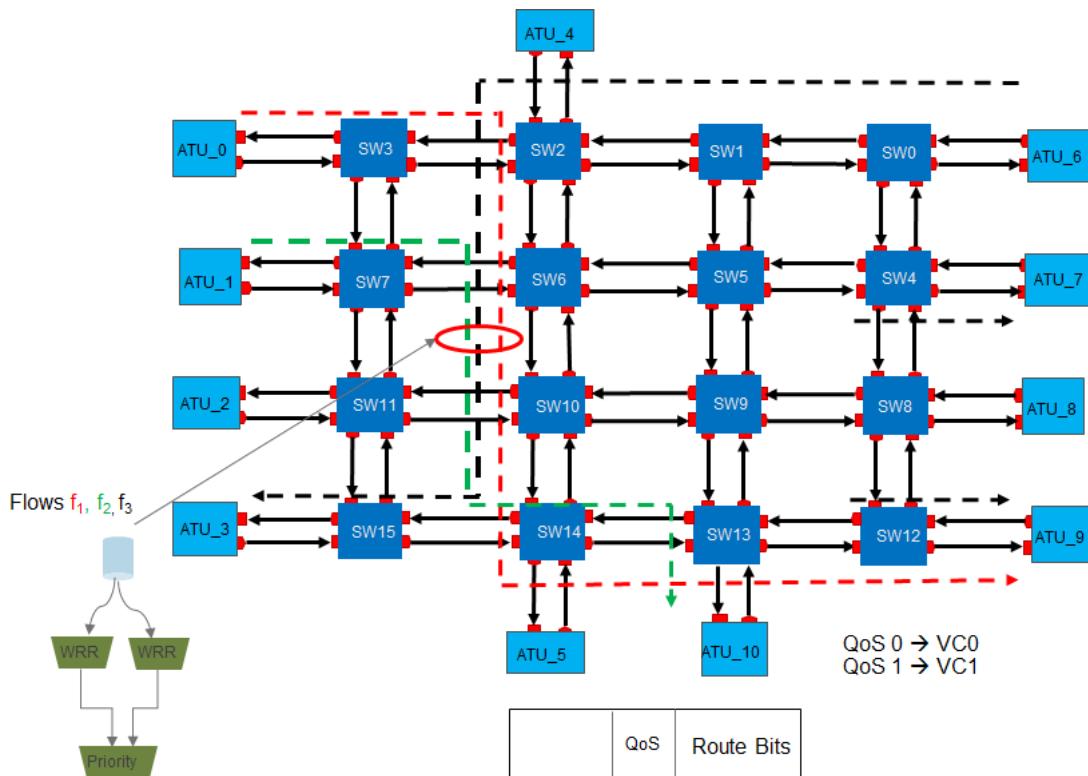
Row No	Number of VCs	Mapping of Flows	Scheduling Schemes	Flow Priority	QoS Values
1	2	<ul style="list-style-type: none"> <li>• Latency Sensitive + Latency bound mapped to VC0</li> <li>• Bandwidth Sensitive + BE flows mapped to VC1</li> </ul>	<ul style="list-style-type: none"> <li>• Input arbiter for VC0 = Weighted Round Robin arbiter</li> <li>• Input arbiter for VC1 = Weighted Round Robin</li> <li>• Master Arbiter = Priority</li> </ul>	<ul style="list-style-type: none"> <li>• Latency Sensitive = P0</li> <li>• Latency Bound = P1</li> <li>• Bandwidth Sensitive = P2</li> <li>• BE Flows = P3</li> </ul>	<ul style="list-style-type: none"> <li>• Latency sensitive + Latency bounded = QoS 0</li> <li>• Bandwidth Sensitive + BE flow = QoS 2</li> </ul>
2	3	<ul style="list-style-type: none"> <li>• Latency Sensitive mapped to VC0</li> <li>• Latency bound mapped to VC1</li> <li>• Bandwidth Sensitive + BE flows mapped to VC2</li> </ul>	<ul style="list-style-type: none"> <li>• Input arbiter for VC0 = Weighted RR</li> <li>• Input arbiter for VC1 = Weighted RR</li> <li>• Input arbiter for VC2 = WRR</li> <li>• Master Arbiter = Priority</li> </ul>	<ul style="list-style-type: none"> <li>• Latency Sensitive = P0</li> <li>• Latency Bound = P1</li> <li>• Bandwidth Sensitive = P2</li> <li>• BE Flows = P3</li> </ul>	<ul style="list-style-type: none"> <li>• Latency sensitive = QoS 0</li> <li>• Latency bound = QoS 1</li> <li>• Bandwidth Sensitive + BE flow = QoS 2</li> </ul>

**Table 13-2 (Continued)VC mapping table for 4-Traffic classes**

<b>Row No</b>	<b>Number of VCs</b>	<b>Mapping of Flows</b>	<b>Scheduling Schemes</b>	<b>Flow Priority</b>	<b>QoS Values</b>
3	4	<ul style="list-style-type: none"> <li>• Latency Sensitive mapped to VC0</li> <li>• Latency bound mapped to VC1</li> <li>• Bandwidth Sensitive mapped to VC2</li> <li>• BE flows mapped to VC3</li> </ul>	<ul style="list-style-type: none"> <li>• Input arbiter for VC0 = Weighted RR</li> <li>• Input arbiter for VC1 = Weighted RR</li> <li>• Input arbiter for VC2 = WRR</li> <li>• Input arbiter for VC3 = WRR</li> <li>• Master Arbiter = Priority</li> </ul>	<ul style="list-style-type: none"> <li>• Latency Sensitive = P0</li> <li>• Latency Bound = P1</li> <li>• Bandwidth Sensitive = P2</li> <li>• BE Flows = P3</li> </ul>	<ul style="list-style-type: none"> <li>• Latency sensitive = QoS 0</li> <li>• Latency bound = QoS 1</li> <li>• Bandwidth Sensitive = QoS 2</li> <li>• BE flow = QoS 3</li> </ul>
4	2	<ul style="list-style-type: none"> <li>• Two Traffic Classes: map each to a different VC.</li> </ul>	<ul style="list-style-type: none"> <li>• Input arbiter for VC0 = WRR</li> <li>• Input arbiter for VC1 = WRR</li> <li>• If one or both are latency sensitive or latency bound, then Master Arbiter = Priority</li> <li>• If both are bandwidth sensitive or BE flows, then Master Arbiter = WRR</li> </ul>	<ul style="list-style-type: none"> <li>• See Table 13-3</li> </ul>	<ul style="list-style-type: none"> <li>•</li> </ul>

A slightly different mapping can also be done using WRR scheduler instead of the priority scheduler on the Master Arbiter. Notice that we have used WRR arbiter at the input arbiter for the VCs. The reason why this is a better choice, is to be able to partition the bandwidth in a work conserving manner at different arbitration points and in the proportion of the flows. If we use a Round Robin algorithm than in a tree network (every flow follows a tree network) the flow entering the tree at the top of the tree gets the least bandwidth. Using dynamic priority to maintain bandwidth can result in oscillations and are not desirable.

Consider the example show below in Figure 13-14. Flows f<sub>3</sub>, f<sub>2</sub> and f<sub>1</sub> are routed using ARTG. ARTG routes



**Figure 13-14 Routing and VC mapping on a Mesh network**

the flows in order of latency sensitivity. That is, flows that are more latency sensitive than others are routed first. The Table 13-4 below shows the route mapping priority based on the classification of the flows.

**Table 13-3 Route mapping priority based on Flow Classification**

Flow Classification	Routing Priority
Latency Sensitive	P0
Latency Bound	P1
Bandwidth Sensitive	P2
Best Effort	P3

Once ARTG maps the flows according to the priority in Table 13-3, it ensures that high priority flows will be routed on the shortest possible path from source to destination. Coming back to the example shown in Figure 13-14, flows  $f_1$  and  $f_3$  are latency sensitive flows whereas flow  $f_2$  is a bandwidth sensitive flow. Since there are only two kinds of flows, based on the recommendation provided in Table 13-2, only 2 VCs would be created. Flows  $f_1$  and  $f_3$  will be given a QoS value of 0 and flow  $f_2$  will be given the QoS value of 1. QoS 0 will be mapped to VC0 and QoS 1 will be mapped to VC1. Following the recommendation provided in Table 13-2, the Master arbiter is a priority arbiter and the input VC arbiter a WRR arbiter. Other flows with similar characteristics would be mapped in a similar manner. The value of the WRR arbiter would be adjusted to make sure that flow get their respective share of the bandwidth.

Note: Multicast flows may be carried in a separate VC.

The above discussion is about how Maestro and Symphony will map flows to VCs when the flows are characterized as priority flows. Flows may also be characterized by the users. For example, the user may

define the criteria that all reads from an Initiator are mapped to different VC than writes (of course customer has make sure that they barrier for overlapping addresses). In this case the flow mapping would be based on the type of transaction.

In the user defined case where other than QoS field is used to determine the VC the flow will be mapped to, Maestro will have to map customer defined field to the QoS field (Symphony supports up to 16 different QoS values). The QoS field in this case will be mapped to VCs.

### **13.13.1ADM, ARTG and User View:**

The User view should enable classification of flows as described above. The user informs the system whether the flow is a Latency sensitive, Latency bound, Bandwidth Sensitive or Best Effort. Once the user classifies the flow, ADM will assign a priority to that flow based on Table 13-3. The ADM flow description already has the data structure to assign a priority to the flow.

The ARTG will extract the flow info from the ADM, which will include among other attributes the priority attribute of the flow. ARTG will use that priority info about the flow to route the flow. Flows which have higher priority will be mapped ahead of those that have lower priority. In this way, the latency sensitive flows have a higher probability to get the shortest path from Src to Dest. ARTG routes the flows and returns the routing information to ADM.

The following information needs to be captured by ADM:

- a. Flow classification and the associated priority

\_\_\_\_\_ Note \_\_\_\_\_  
Flow <-> VC association  
\_\_\_\_\_

\_\_\_\_\_ Note \_\_\_\_\_  
Bandwidth associated with a VC and the link  
\_\_\_\_\_

- b. Breakdown of the traffic allocated to the egress VC that is coming from different input ports

ADM should be able to calculate all of the above, as it has the route information for each flow, VC mapping information and nodes through which the flows pass. From (c) and (d) above, the system would be able to assign the weights and priority on the Input VC arbiter and the Master arbiter on the egress port.

### **13.13.2TCL Commands and Rules**

TCL commands are needed to perform the following functions:

1. Max number of VCs for QoS purposes in the system: TCL command for the user to put in the number of VCs they would like to allow in the Network. Note Maestro would instantiate the VCs where needed, but would not exceed the maximum specified number. Maestro will map flows into the VCs based on the default mapping rules in ADM or ones specified by the user.
  - a. "Max\_QoS\_VCs"=[Number\_of\_VCs]
2. Mapping Rules: There could be two sets of rules. One set of rules are default rules that are pre-defined in Maestro. If the user does not want the default rules to be used they can specify their own.
  - a. TCL command for default rules: "TrafficClass\_to\_VC\_Mapping: Default".
  - b. TCL command for user defined rules: "TrafficClass\_to\_VC\_Mapping: Custom"

3. If “TrafficClass\_to\_VC\_Mapping: Custom” == True:
  - a. “Latency\_Sensitive\_Flows\_Mapped\_to”: [VC No]
  - b. “Latency\_Bound\_Flows\_Mapped\_to”: [VC\_No]”
  - c. “Bandwdith\_Sensitive\_Flows\_Mapped\_to”:[VC\_No]
  - d. “Best\_Effort\_Flows\_Mapped\_to”:[VC\_No]

e. Note: Raise an objection if the number of VCs specified in this step is more than the “Max\_QoS\_VC”.
4. If “TrafficClass\_to\_VC\_Mapping: Custom” == True:
  - a. “VC\_Arbiter\_Setting” = [Default, Custom]
5. If “VC\_Arbiter\_Setting=Custom” == True
  - a. “Set\_input\_VC\_Arbiter” = [RR, WRR, PRI]
  - b. “Set\_Master\_VC\_Arbiter”=[RR, WRR, PRI]

Default rules:

Based on Table 13-2, the rules are as follows:

Rule 1: If two VCs and four Traffic Classes, use row 1 setting.

Rule 2: If three VCs and three Traffic Classes, map each traffic class into a VC.

Rule 3: If three VCs and four Traffic Classes, use row 2 setting.

Rule 4: If four VCs and four Traffic Classes, use row 3 setting.

Rule 5: If two VCs and two Traffic Classes, use row 4 setting.

Custom rules are global and are based on QoS class mapping.

### 13.13.3 GUI Representation of VCs

VCs be color coded, based on the class of service they carry. If a VC carries two or more classes then the color of the VC should be the color corresponding to the higher QoS class.

## 13.14 Effective Bandwidth

Effective bandwidth is defined as the actual bandwidth that should be allocated on the link to meet the bandwidth and latency requirements of the flow.

Effective bandwidth of the flow also depends on the packet header style, that is, whether the packet style is header parallel, or header serial.

For the header serial (header and data are transmitted on the same set of wires) mode, the width of the link is determined by the max (data beat, header size). If the header size is greater than the data beat size, then the link width will be dominated by the header width. In which case the inefficiency needs to be added to the link when the flow is mapped on to it. This inefficiency needs to be calculated for each link.

Traffic generated from the end points in the SoC is rarely ever well behaving, in the sense that it is bursty in nature and not TDM (time division multiplexed) like traffic. The bursty nature of the traffic dictates that, at times, the Initiator will send back to back transactions causing the peak rate to the flow to reach the maximum link transmission rate, while the average bandwidth would be much lower. The question that has to be answered, is which bandwidth number does the system use for the mapping the flow - the peak rate or the average. The customer may provide both or just the one.

According to queuing theory, if we use the average bandwidth to map flows on the links, and the links occupancy is close to 100%, the system risks going into severe congestion increasing the latency of each transaction. If the peak bandwidth is used for mapping flows, then the link utilizations would be low and will increase the overall cost of the solution.

Let us introduce a multiplier  $\alpha$  (alpha). The multiplier augments the average bandwidth by the factor  $\alpha$ . The value of this multiplier depends on the classification of the flow. For P0 flows, the value of  $\alpha$  would be higher than lower priority flows. The value of  $\alpha$  would be determined by performance simulations.

# 14

# Virtual Networks in Symphony

This section describes virtual network (VN) support in Symphony. It lists the requirements for VNs, describes how VN capability is architected in Symphony, and presents some examples of VNs in Symphony.

## 14.1 Symphony VN Definition

A virtual network, from a Symphony perspective, is a connected subset of the interconnect. Symphony VNs are architected to be independent of each other. The VNs behave independently in that flows on one VN do not impact flows on other VNs and that they each can be provisioned to provide different QoS levels to the external devices attached to them.

There are several motivations for using VNs:

- ▶ “Enables the prevention of congestion between traffic flows in the system, by enabling the system designer to separate masters with conflicting requirements on to different VNs. For example, high-bandwidth bus traffic sources can be separated to prevent blocking the flow of latency critical bus traffic” -- ARM QVN Spec
- ▶ Protecting control plane traffic from data plane traffic on an SoC that supports both control plane processing as well as data plane processing on the same SoC
- ▶ Allowing trusted and non-trusted domains on the same SoC

## 14.2 VN Requirements

Symphony VNs are required to support the following:

- ▶ Support a configurable number of VNs in a NoC instance. The Symphony architecture does not limit the number of VNs in a NoC – however, 16 VNs in a NoC will be a practical limit. Note that ARM may architecturally support up to 16 VNs in the system – a Symphony NoC (or connected NoCs) will support up to 8 VNs.
- ▶ Support a maximum of 4 VNs on any single master or slave
- ▶ Support independent levels of service (QoS) for each VN in the NoC
- ▶ Flexible mapping of requests to VN queues
- ▶ External masters and slaves that implement the ARM Quality of Service VN Specification (QVN – Rev. r1p0). This includes supporting the token management interface detailed in the specification.
- ▶ Support the following VN systems:

- ◆ Multiple VN Masters and a single slave (n:1)
- ◆ Single Master and single Slave (1:1)
- ◆ m VN Masters and n VM Slaves (m:n) given that the ATUs connected to the slaves manage credits independent of each other
- ▶ Support hybrid applications:
  - ◆ Mixture of VN and non-VN capable masters
  - ◆ Mixture of VN and non-VN capable slaves
- ▶ Protection of resources allocated on the interconnect (bandwidth, buffers, etc.) to each Virtual Network. That is, one virtual network should not be allowed to unfairly utilize resources of the interconnect which were allocated to another virtual network.
- ▶ Prevent a denial-of-service attack. A misbehaving virtual network should not be able to prevent other virtual networks in the system from functioning.
- ▶ Performance monitoring, trace generation, and firewall functions should be able to be implemented on a virtual network basis
- ▶ Performance monitoring/debug probes will be VN aware
- ▶ In VN aware mode, one VN will not be able to log/trigger/trace events/links based on other VN traffic

The following restriction exists on VN topologies:

- ▶ Symphony will not support a topology where there are “m” VN-capable masters and “n” VN-capable slaves ( $n > 1$ ) where the slaves are required to coordinate credit management collectively. An example of this is where there are two QVN slaves and Symphony is configured to be in QVN pass-through mode (see Section Pass-through Mode).
- ▶ In QVN, only one VN is supported on an external interface if the interface is configured to be AHB or APB

## 14.3 Symphony Architecture for VNs

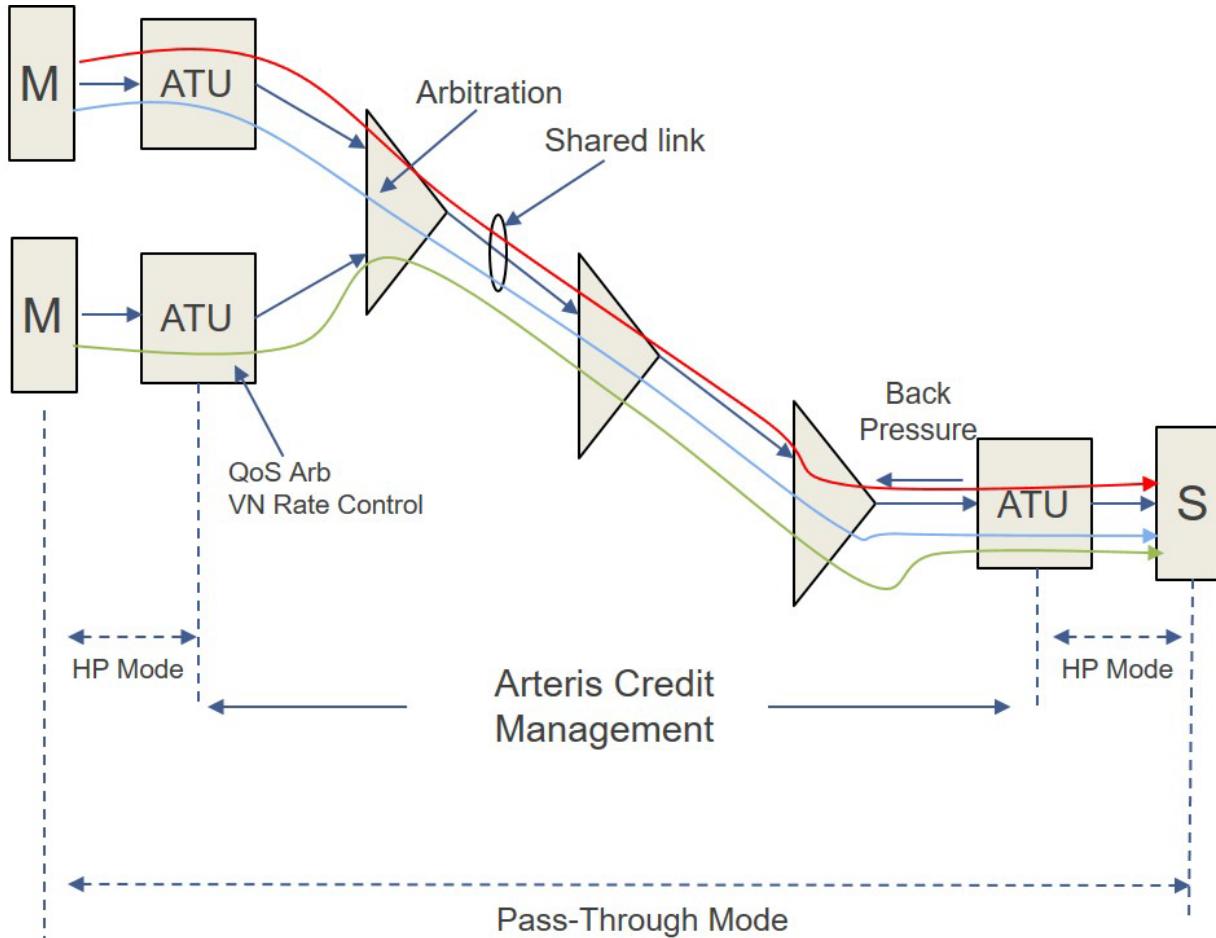
The major elements in Symphony used in VNs are:

- ▶ Per VN queues in initiator and target ATUs
- ▶ Methods to map from external interface “flows” to VNs internal to the NoC and back again
- ▶ Credit-based end-to-end flow control for each VN
- ▶ Per-VN scheduling algorithms
- ▶ Virtual Channels (VCs) to provide links in the interconnect for the VNs
- ▶ Flexible mapping from VNs to the VC resources used to implement the VNs
- ▶ The Symphony packet protocol is VN aware – credit messages have a VN field, so credit can be requested/granted on a per VN basis (see Section Arteris Transport Packet Protocol on page 35 for more details on the Symphony packet protocol)

## 14.4 Virtual Channels (VCs) as Resources for Creating VNs

The Symphony architecture uses VCs as resources to build VNs. It's possible to build a multi-VN interconnect with no VCs – this is shown in Symphony VN interconnect with No VCs.

**Figure 14-1 Symphony VN interconnect with No VCs**



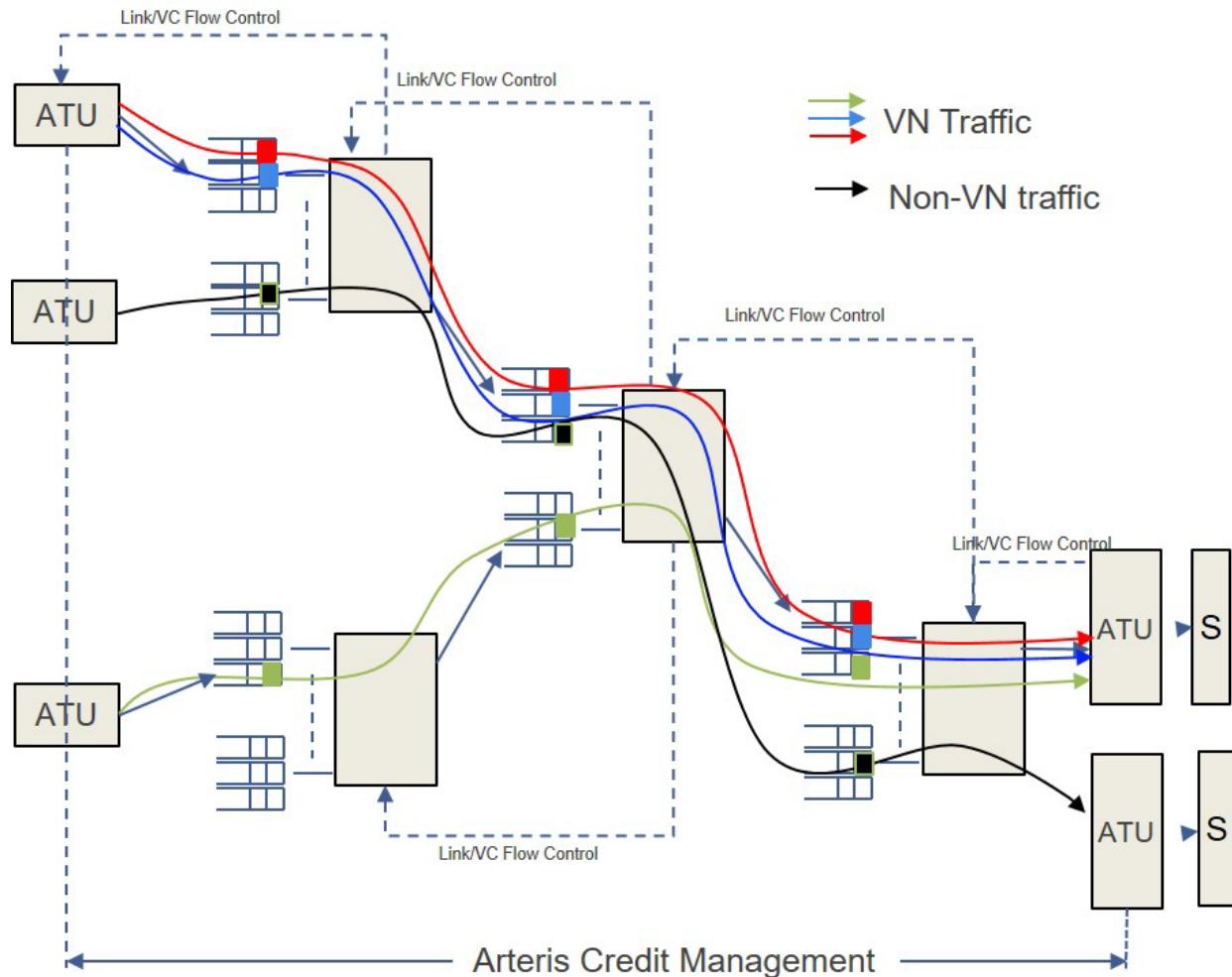
For this network, the following statements apply:

- ▶ By definition, Virtual Channels are not available to isolate traffic in the Transport Network
- ▶ Symphony will use Source routing (pre-determined paths) to isolate VN traffic where possible
  - ◆ VN traffic is regulated traffic while other traffic may not be as regulated
- ▶ The ATU will regulate the injection rate using the following:
  - ◆ Arbitration algorithm to resolve local contention on VN/QoS basis
  - ◆ Rate limiting at the VN/QoS level

- ◆ Credit management to guarantee acceptance at the target ATU for the VN and QoS
- ▶ Bounded delay and bandwidth
- ◆ Arb algorithms + Credit Management

In contrast to the VN NoC shown in Symphony VN interconnect with No VCs, a VN interconnect can be built using VCs as resources. This is shown in Symphony VN interconnect with VCs.

**Figure 14-2 Symphony VN interconnect with VCs**



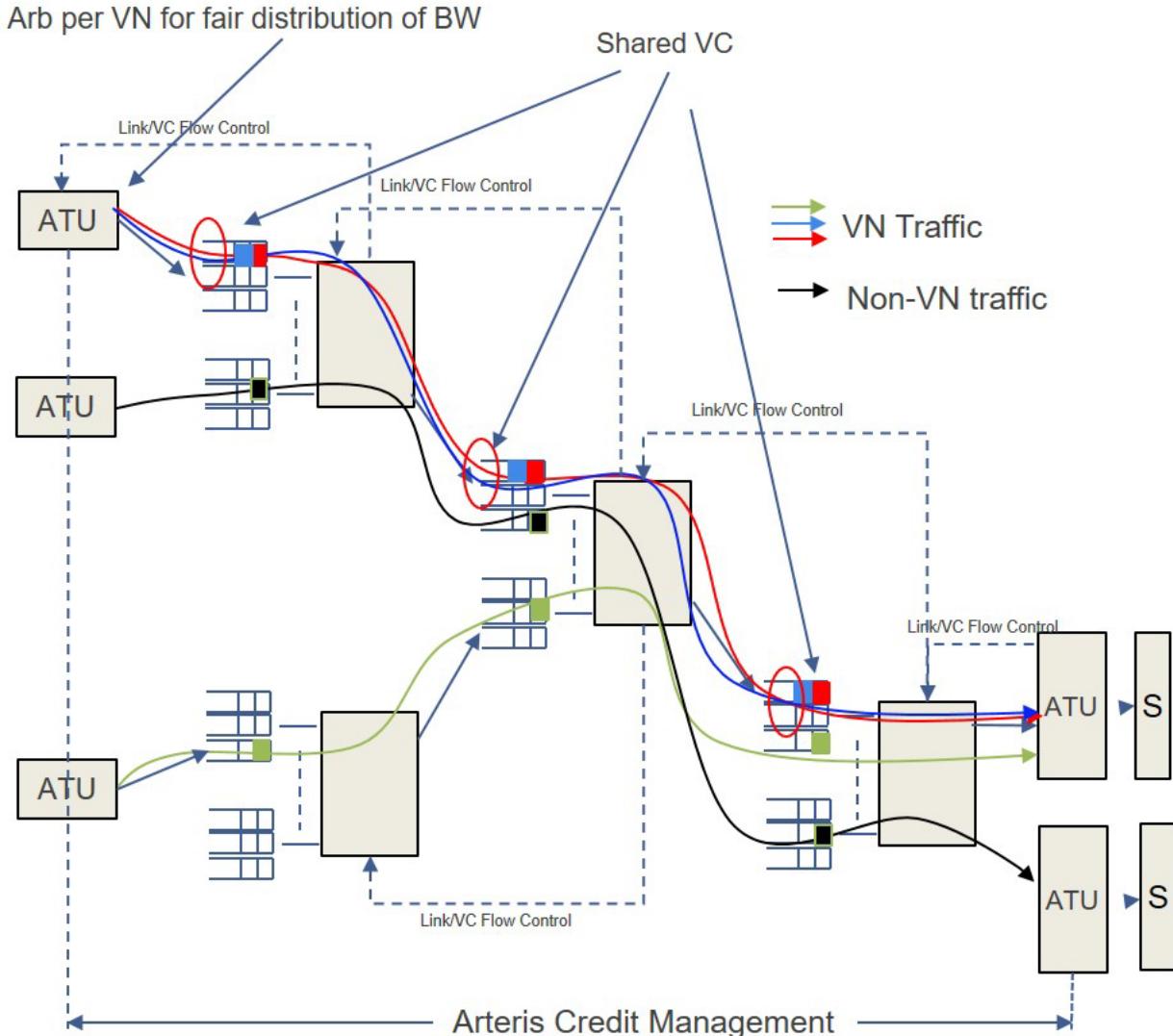
In the VN interconnect built with VCs, the following statements apply:

- ▶ Extensive use of Virtual Channels to support QVNs
- ▶ Limit of 16 VN on the physical channel
- ▶ Asymmetric VC allocation on the ports
  - ◆ Not all ports will have the same number of VCs
- ▶ ARM currently limits 8 Virtual Networks per physical channel
- ▶ Back pressure support per VC
- ▶ VNs mapped to VCs

- ▶ Fair Queuing arbitration at switches and ATUS and Credit Management at the ATUs ensure that each VN gets its fair share of NoC resources
- ▶ VC queuing isolates VNs and Non-VN traffic

VNs can be created that share VCs. An illustration of this is shown in Symphony VN interconnect with shared VCs.

**Figure 14-3 Symphony VN interconnect with shared VCs**



The following statements apply to Symphony VN interconnect with shared VCs:

- ▶ Behavior:
  - ◆ VCs are assigned on a packet basis and not Phit basis
  - ◆ Phits can be interleaved on the channel (physical link), when they are going to a different VC
  - ◆ Phits belonging to different packets cannot interleave in the downstream VC
  - ◆ Switch Arbitrates for an output port:

- ✓ Flow-through switch – Whenever the input port becomes valid and port is not allocated
- ✓ Pipeline switch – every Phit to allow for scenarios where the VC backpressure prevents forward progress or to enable a high priority packet to flow through. Note the VC is not shared.
- ▶ Shared Virtual Channels:
  - ◆ VNs can be grouped together if they are destined for the same Target and have the same QoS requirement BUT only if they have separate Credit control
  - ◆ Bandwidth allocated at each Switch to the Shared VC
  - ◆  $\checkmark_B = \sum_{k=0}^{n-1} B_k$ ,  $n = \# \text{ of } VNs \text{ sharing the VC}$
  - ◆ Fair Queuing + credit regulation will ensure a bounded delay in the system

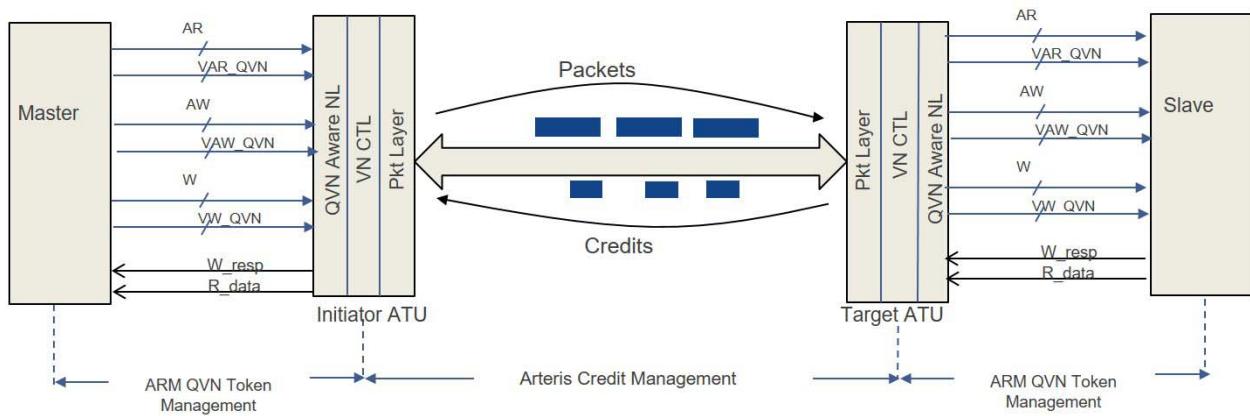
## 14.5 Symphony QVN Support

The use of the term QVN implies that there are both initiators and targets attached to the Symphony interconnect that use the ARM QVN protocol. The QVN protocol uses tokens to control the flow of data on any given VN. QVN ensures that a transaction is not issued across an interface unless there is space allocated for it in the destination. The Symphony interconnect sits between initiators (which request tokens) and the targets (which grant tokens). There are two modes in Symphony that can be used to manage token passing between initiators and targets.

### 14.5.1 High Performance Mode

In high performance (HP) mode, flow control based on tokens (QVN method) and credits (Symphony method) is broken up into three regions as is shown in QVN High Performance Mode.

**Figure 14-4 QVN High Performance Mode**



This is done as follows:

- ▶ Symphony terminates the QVN interfaces at the initiator and target ATU
- ▶ The Initiator and Target ATUs handshake with the Master and Slave respectively to manage tokens

- ▶ Master will not issue a transaction unless it has been granted a token by the Initiator ATU
- ▶ The target ATU will not transfer the transaction to the Slave unless it has been granted a token by the Slave
- ▶ Transfers between the initiator and target ATUs in the interconnect are controlled by credits
- ▶ The initiator ATU will not send a transaction to the target ATU unless it has credits from that ATU
- ▶ Credits can be pre-allocated to improve ramp up time
- ▶ Symphony will map ARM QVN on to Symphony VN Service

In a credit management system, system throughput can be limited by how fast the credits are provided to the initiator. For an initiator  $i$ :

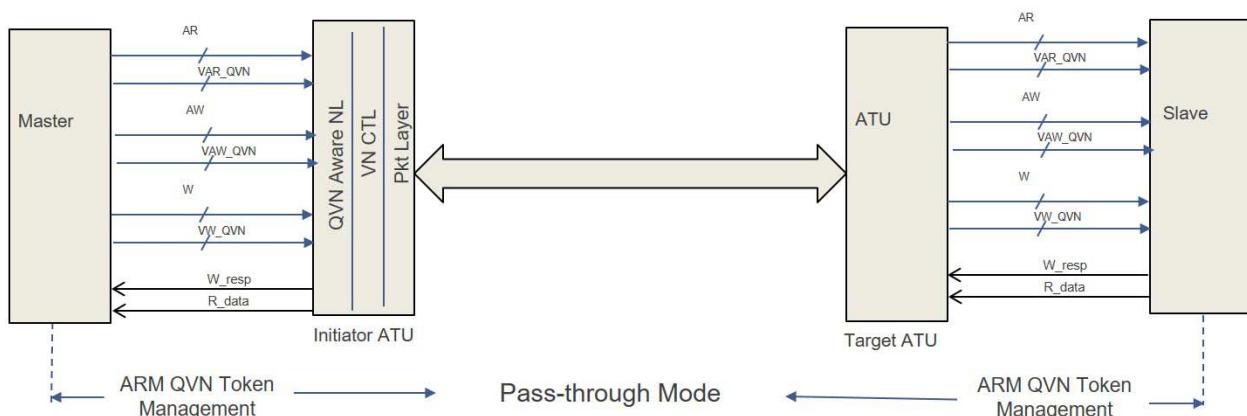
$$\text{Bandwidth}_i = ci * \frac{1}{rtt}, \text{ where } rtt \text{ is roundtrip time; } Ci \text{ credits assigned}$$

- ▶ In the ATU-ATU credit management system, the number of tokens allocated to each Initiator ATU can be adjusted by Arteris tools, based on the bandwidth requirements
- ▶ Adjusting tokens at the QVN Master and Slave would be handled by the SoC driver layer and not by Arteris
- ▶ Buffering Requirements in the Target ATU depend on the number of credits that the Target can issue and its input and output rates
- ▶ The buffering requirements would be minimal if the Target ATU input and output rates (target ATU drain rate to the Slave) are matched

## 14.5.2 Pass-through Mode

In pass-through mode, flow control based on tokens (QVN method) and credits (Symphony method) is broken up into two regions as is shown in QVN Pass-through mode.

**Figure 14-5 QVN Pass-through mode**



This is done as follows:

- ▶ Initiator and Target ATU do not manage credits on behalf of the Master/Slave
- ▶ Initiator and Target ATUs translates QVN token request/response into Arteris credit packets and back into QVN token request/response
- ▶ Master will not issue a transaction unless it has a token available

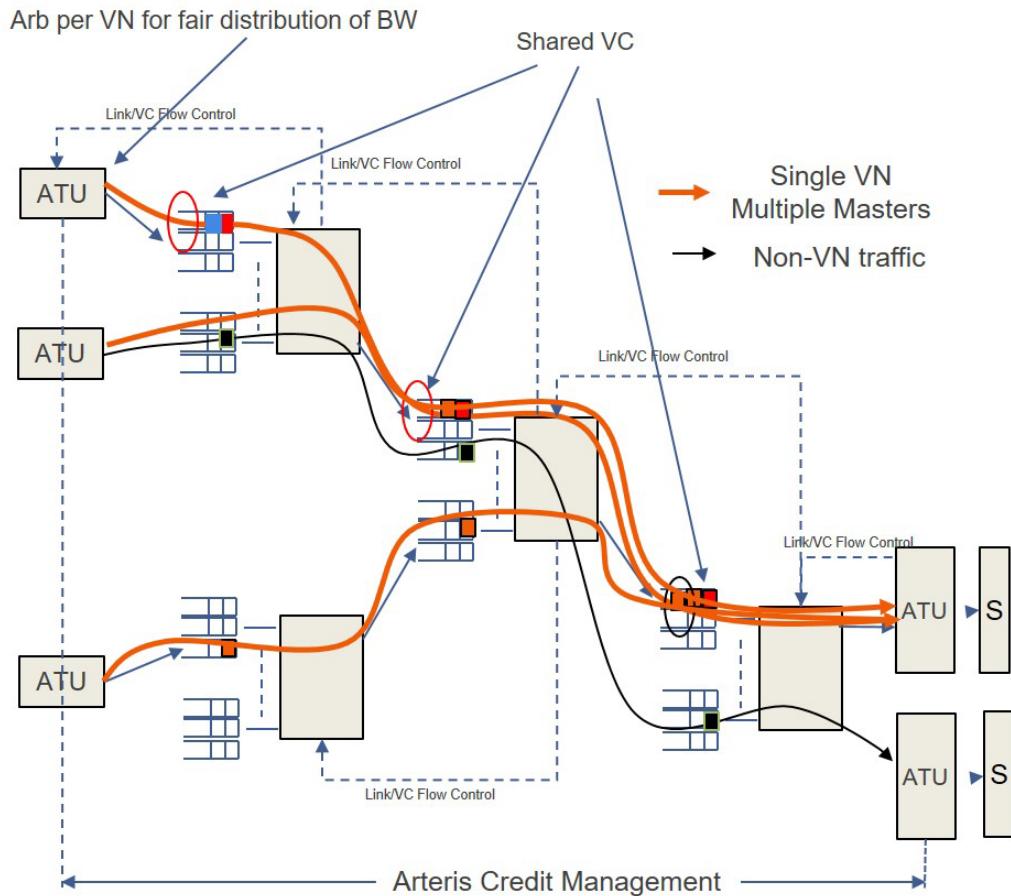
Notes on pass-through mode:

- ▶ The performance can potentially be lower compared to high performance mode
- ▶ Longer round trip time required
- ▶ Performance proportional to number of credits allocated and roundtrip time
- ▶ Could optimize credit allocation if within the Arteris system
- ▶ Buffering requirements at the Target ATU:
  - ♦ If the following two conditions are met, Then buffering requirements would be minimized (1 beat of buffering/VN):
    - ✓  $\sigma_{Peak\_ATU} \leq \sigma_{Peak\_Slave}$  where  $\sigma$  is the input rate
    - ✓ Always buffer space available for a transaction with a token at the slave

### 14.5.3QVN Example Comparing HP to Pass-through Modes

VN interconnect with N VN Masters and 1 VN Slave details a interconnect with multiple VN masters, one VN slave, and one non-VN slave. The following is a comparison of the two modes:

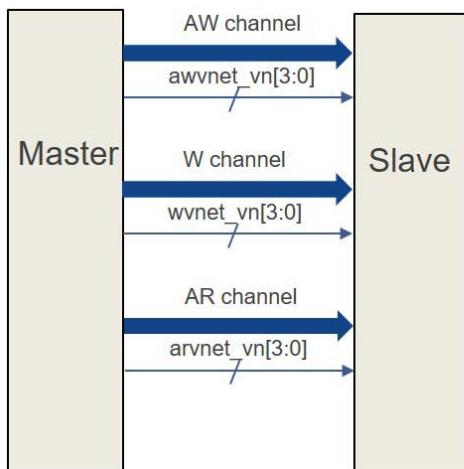
- ▶ Pass-Through Mode:
  - ♦ In the pass-through mode, only the Masters/slave manage credits
  - ♦ The Target ATU does nothing other than pass the transactions and the packets through
  - ♦ In the pass-through mode, both the Initiator ATU and the Target ATU do not manage credits
- ▶ HP Mode:
  - ♦ VN credit is terminated at the Initiator and the Target ATU
  - ♦ Target ATU manages credits for the VN
  - ♦ Target ATU will issue credits to the VN Masters based on the configured bandwidth requirements
  - ♦ VN traffic from multiple masters will share the VC where possible

**Figure 14-6 VN interconnect with N VN Masters and 1 VN Slave**

## 14.6 ARM QVN Impact on External Interfaces

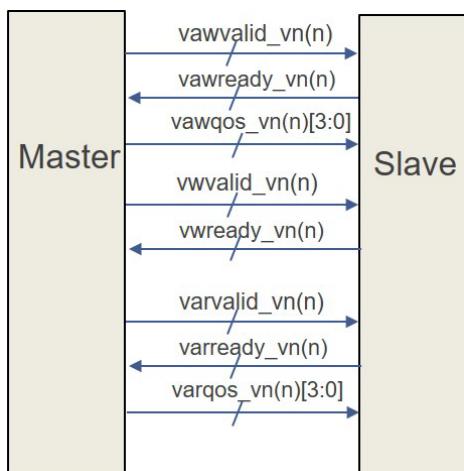
The QVN specification impacts external AXI interfaces:

- ▶ Defined as an extension for AXI-3 and AXI-4 protocols
- ▶ Impacts the protocol in two ways:
  - ◆ The three forward direction channels (read request, write request, and write data) each have a bit vector that identifies what VN the transaction is for so that the slave knows what VN the transaction is for. These signals are shown in QVN Signals on AXI read request, write request, and write data channels for a case where there are 4 virtual networks (hence the bit vector width is 4).

**Figure 14-7 QVN Signals on AXI read request, write request, and write data channels**

- AW is write request channel, awvnet\_vn specifies what virtual channel request is for
- W is write data channel, wvnet\_vn specifies what virtual channel write data is for
- AR is read request channel, arvnet\_vn specifies what virtual channel request is for

- ▶ Separate token management channels for write requests, read requests, and write data on a per virtual channel basis – see QVN Token Management Channels.
  - ◆ Each virtual network has its own token management channels – if there are four virtual networks, there are 4 sets of token management channels
  - ◆ On a given VN, there are separate token management channels for write requests, read requests, and write data
  - ◆ The token management channels for read and write requests have valid, ready, and QoS signals whereas the token management channel for write data only has valid and ready and no QoS
  - ◆ Token management channels are independent of other channels – so token requests/responses can occur at the same time as request and data transfers
  - ◆ Given the number of token channels, there is a large amount of token handling BW

**Figure 14-8 QVN Token Management Channels**

- Example – master and slave support four virtual networks:
  - 12 total token management channels
  - Write request token channels:  
 $4 * (\text{valid} + \text{ready} + \text{QoS}) = 24$  signals
  - Write data token channels:  
 $4 * (\text{valid} + \text{ready}) = 8$  signals
  - Read request token channels:  
 $4 * (\text{valid} + \text{ready} + \text{QoS}) = 24$  signals

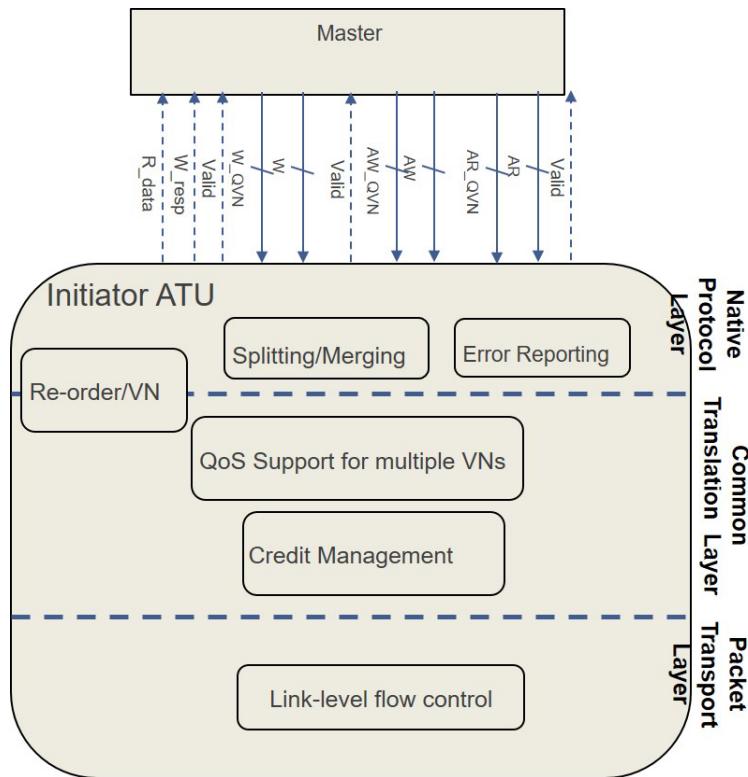
## 14.7 ATUs and VNs

The ATUs in Symphony play a major role in supporting the VN function. This section highlights functionality in the ATU that is used for VNs. For a more detailed description of the ATU, see Section Virtual Networks in Symphony on page 215.

### 14.7.1 Initiator ATU

The initiator ATU receives transactions from the Master. An external master and an initiator ATU are shown in Initiator ATU Support for VNs.

**Figure 14-9 Initiator ATU Support for VNs**



The initiator ATU does the following:

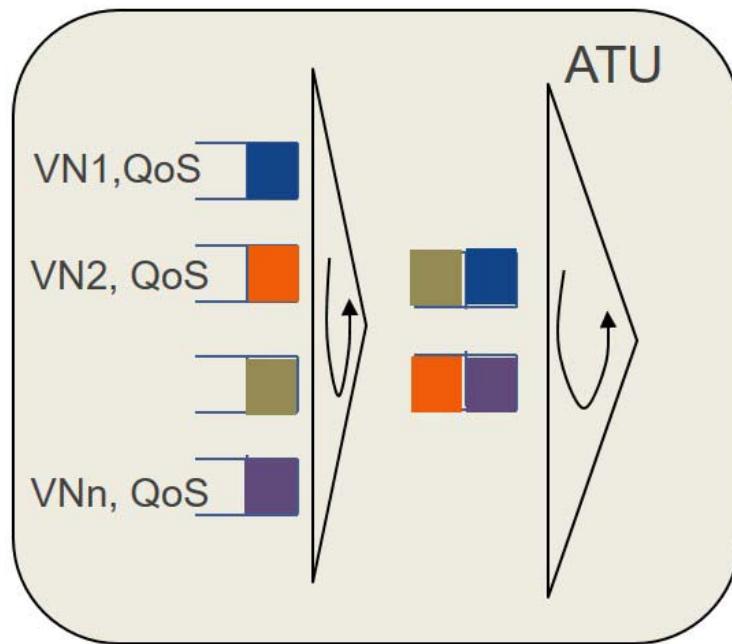
- ▶ ATU native protocol layer terminates the QVN Protocol and extracts the information to be passed to the common translation layer (CTL)
- ▶ The request is split if the transaction size is greater than Max\_pkt size
- ▶ All splitting and merging is done at the Initiator ATU per VN
- ▶ Reordering is done per VN using logical buffering
- ▶ Requests/response mapped to appropriate queues within the CTL depending on the configured VN-Queue mapping
- ▶ Per VN credit management
  - ◆ Credit handshake with the QVN Master/Slave in the case of HP mode
  - ◆ Credit handshake with the Target ATU in the case of HP mode

## 14.7.2 Initiator ATUs, VNs, and QoS

The initiator ATU supports QoS on requests from Masters for VNs via mapping functions to different queues and by using different scheduling algorithms based on the QoS and how the ATU is provisioned. See Virtual Network and QoS in the Initiator ATU.

- ▶ Different mapping options for transactions are available:
  - ◆ Queueing based on QoS value
  - ◆ Queueing based on QVN id
  - ◆ Hierarchical queueing based on:
    - ✓ QVN id
    - ✓ QoS
- ▶ The following scheduling algorithms will be available for VNs:
  - ◆ Priority with Starvation protection
  - ◆ Round Robin
  - ◆ Deficit Round Robin
  - ◆ Peak rate limiting
  - ◆ TDM-like algorithm for Isochronous flows

**Figure 14-10 Virtual Network and QoS in the Initiator ATU**

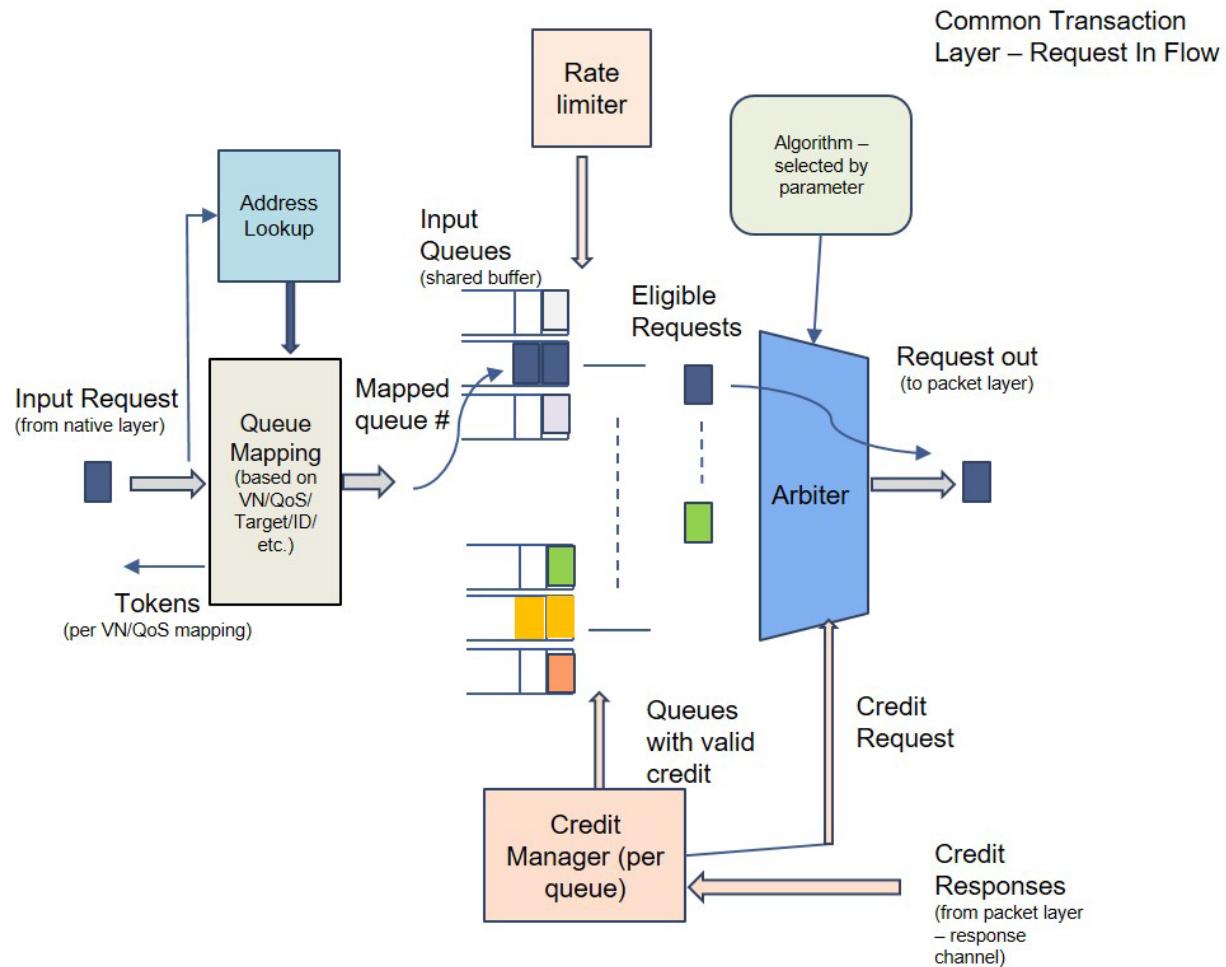


## 14.7.3 Initiator ATU and VN Processing Flow

The initiator ATU begins the handling of VN processing in Symphony. See VN Processing Flow in Initiator ATU for a conceptual view of this processing flow. The steps in this flow include:

- ▶ Input request flow from the Native Layer
- ▶ Address lookup will determine the destination node
- ▶ Queue mapping table maps the incoming requests to the appropriate queue based on the QoS value, VN\_id and the destination node
- ▶ Queues eligible for service if and only if the credit count >0 and it has not exceeded the configured bandwidth rate
- ▶ Link Arbiter arbitrates between all eligible queues for packet transmission

**Figure 14-11 VN Processing Flow in Initiator ATU**

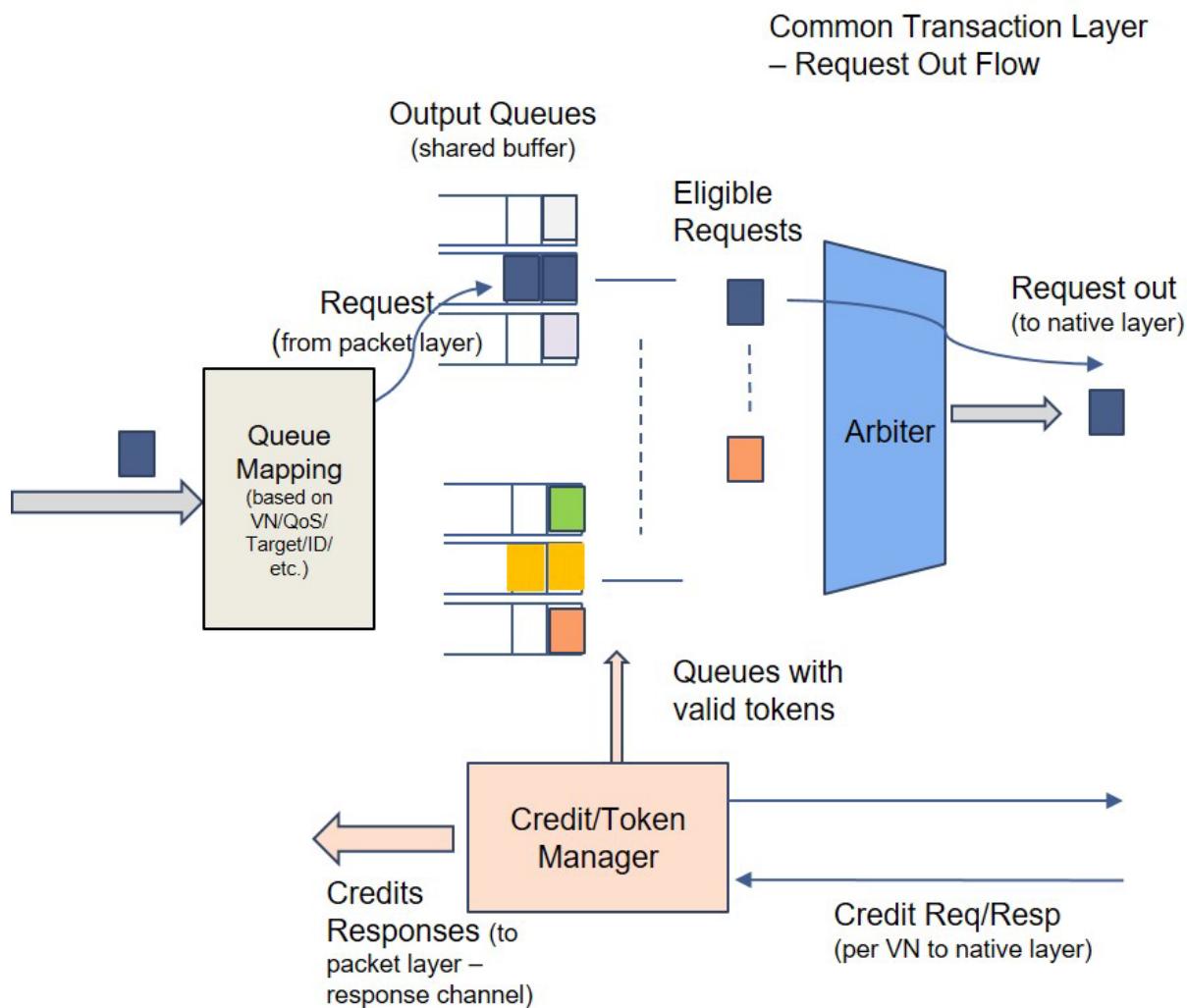


## 14.7.4 Target ATU and VN Processing

The target ATU terminates VNs and processes packets received on one or more VNs. See VN Processing in a Target ATU for a conceptual view of this processing. The processing steps include:

- ▶ Requests are mapped to the appropriate queue based on the output of the queue map per
- ▶ Requests are forwarded to the arbiter only if the queue (VN) has credit/s available from the Slave to forward transactions

- ▶ The arbiter arbitrates all eligible queues for transmission
- ▶ The Slave issues credit per VN to the Target ATU based on local service policy
- ▶ Target ATU will issue a credit to the Initiator ATU based on the QoS policy once the request is dequeued and sent to the slave if the request was from a dedicated (as opposed to shared) request pool
- ▶ Target ATU will issue link level flow control if it has no buffers available

**Figure 14-12 VN Processing in a Target ATU**

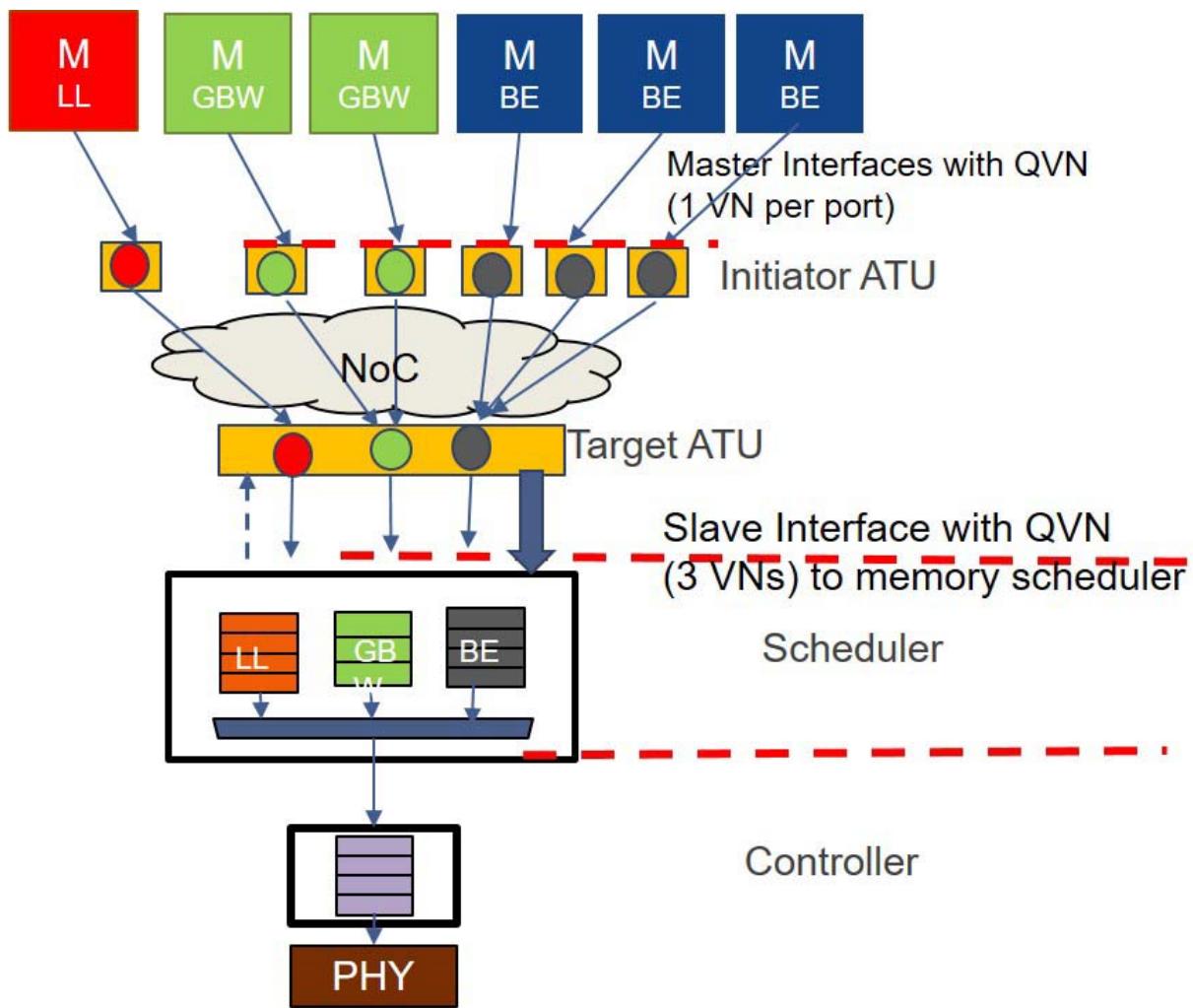
## 14.8 VN Examples

This section contains examples of applications implementing VNs.

## 14.8.1 Memory Controller Example

VN interconnect with Memory Controller shows an example with six masters connected to a memory controller through an interconnect. There are three different QoS requirements among the three masters.

- ▶ Masters:
  - ◆ Three Best Effort Masters
  - ◆ Two Guaranteed BW Masters such as GPUs
  - ◆ One low latency Master such as a CPU
- ▶ Slave:
  - ◆ Memory scheduler or an intelligent memory controller
- ▶ System Config:
  - ◆ 3 VNs established: one for each traffic type
  - ◆ Appropriate VN:QoS mapping in the switches and the Target ATU to meet traffic requirements
  - ◆ QoS aware interface to the memory scheduler
- ▶ Target ATU will forward requests to the scheduler if and only if:
  - ◆ The VN has a credit available
  - ◆ VN wins the arbitration
- ▶ Target ATU will rate regulate masters within the VN based on how it distributes credits between them
- ▶ Note: For Symphony to support this configuration, ARM QVN is not a requirement

**Figure 14-13 VN interconnect with Memory Controller**

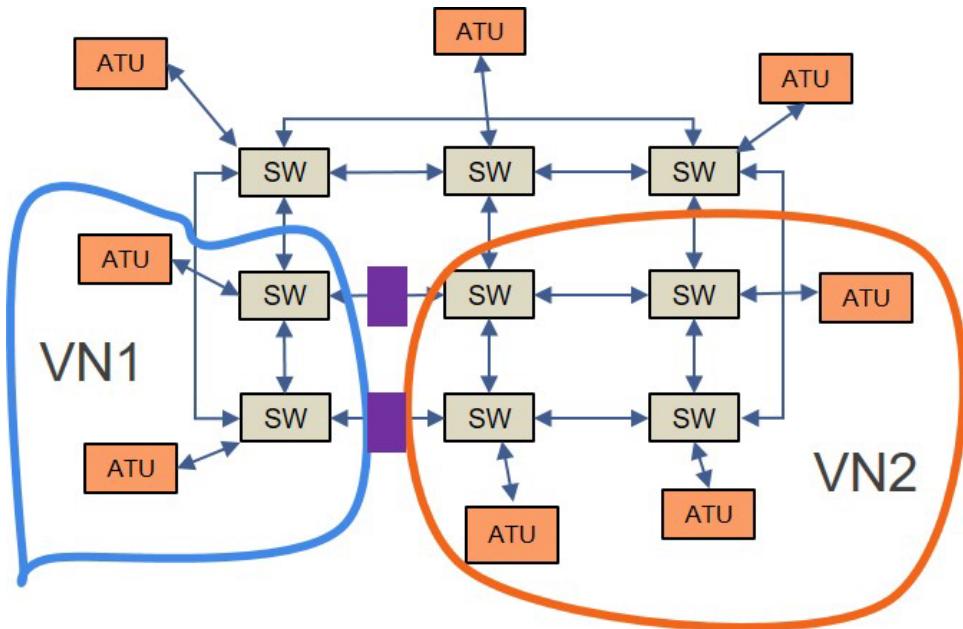
## 14.9 Virtual Network Hard Partitioning

Symphony supports a method to perform a “hard” partitioning function that keeps packets from certain links and switches based on a filtering criteria enforced by a Symphony Firewall. A conceptual diagram of this is shown in Conceptual Diagram of VN Hard Partitioning. The filtering is VN aware are noted below:

- ▶ Packets based on certain fields are not allowed to pass through certain links and switches in the interconnect
- ▶ This is an insurance policy if there is a configuration error or the network is improperly provisioned
- ▶ Firewall will be placed on provisioned links to prevent unwanted traffic from entering the secure part of the network
- ▶ Firewall is “VN” aware and will either:
  - ◆ Mark packet as errored to be discarded at the Target ATU

- ♦ Packet turned around and sent back to the Initiator ATU before it interferes with other VNs

Figure 14-14 Conceptual Diagram of VN Hard Partitioning





# 15

# Configuration

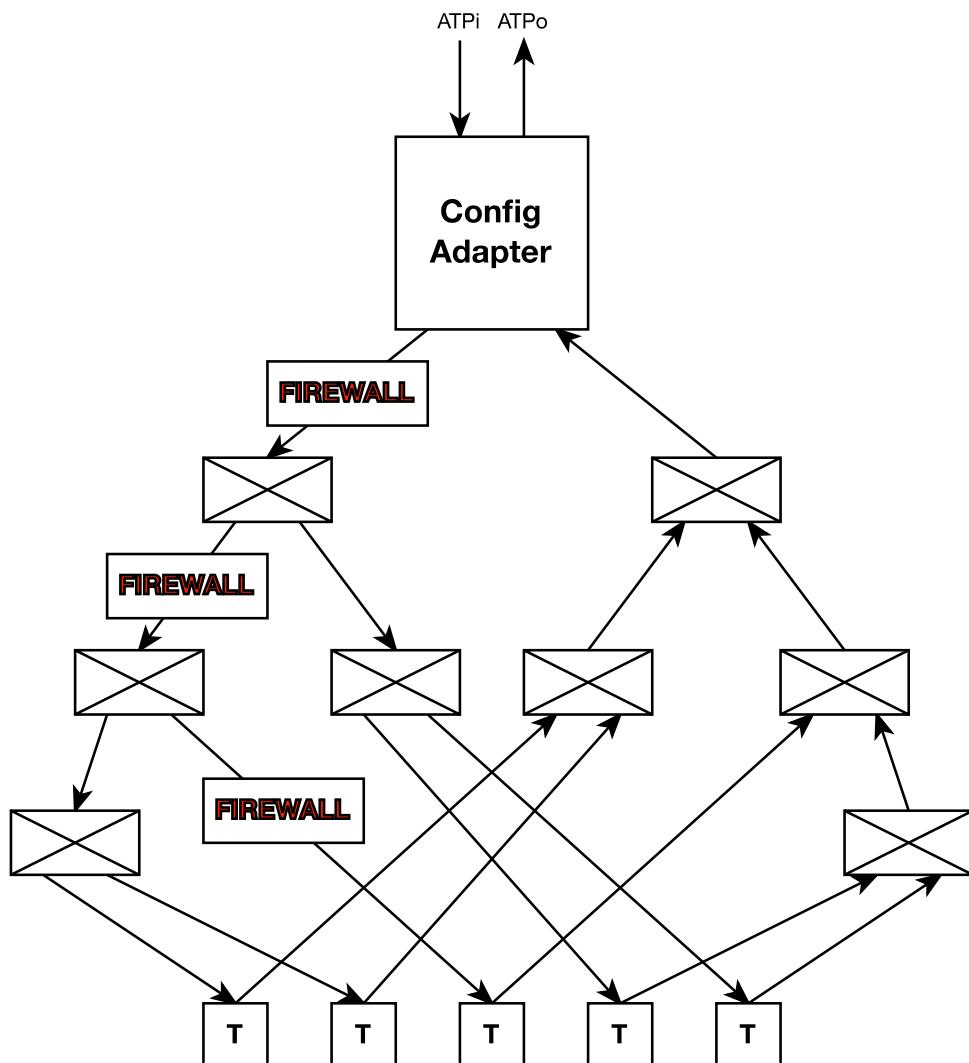
## 15.1 What's considered configuration

1. Any readable and/or writeable register that exists in the interconnect is considered configuration and sits in the configuration address space. Examples of this are:
  - a. Firewall configuration
  - b. Error Status registers and interrupt control registers
  - c. Performance monitoring configuration.
  - d. Debug registers and configuration.
2. Exceptions: If the documentation explicitly states a register sits in another address space. An example of this is the ATU TrustZone Registers which sits in the address space of its ATU.

## 15.2 Configuration Fabric

1. The Configuration fabric will be physically separate from other fabrics in the interconnect.
2. Multiple Configuration fabrics: There can exist multiple configuration fabrics that can be split apart for multiple reasons. The ultimate restriction will be what the software can support, and not the architecture or the RTL.
3. Security in the Configuration fabric will be achieved through the use of firewall IP.
4. The configuration fabric will be programmable to the same extent as any other fabric in the system.

An example configuration fabric is shown in Figure 15-1 on page 234.

**Figure 15-1 The Configuration Fabric with Firewalls.**

## 15.3 Configuration Register Addresses

1. Individual registers will exist in 4 byte or 8 byte boundaries.
2. 8 byte registers must support operations done on 4 byte boundaries.

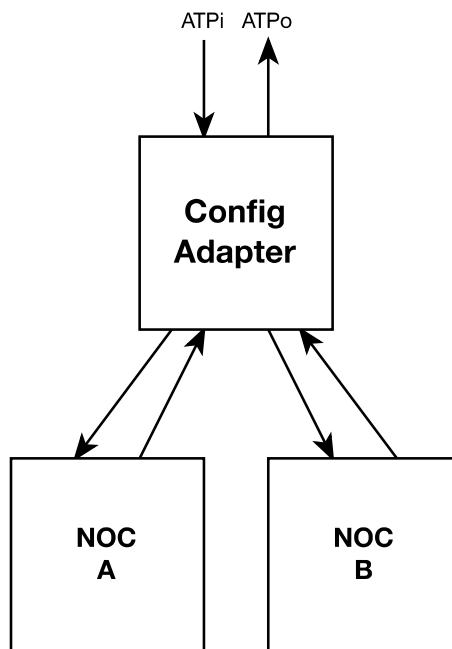
## 15.4 Configuration Register Address Spaces

1. For registers associated with virtual network aware targets, there can be three address regions that are not required to be contiguous in address space:
  - a. Hypervisor Address region.

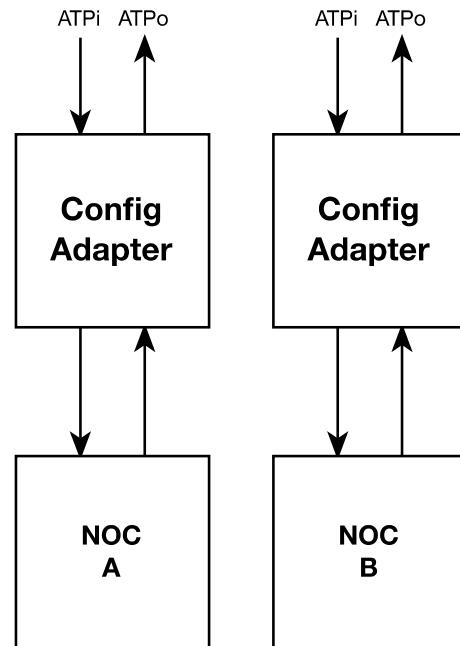
- ♦ Contains all registers associated with a target.
- b. Debug Address region.
- ♦ May contain a subset of registers associated with a target.
- c. User Address region.
- ♦ Contains a subset of registers associate with in target.
- d. Within these three regions are sets of registers that are divided up based on the number of virtual networks a target supports that will have unique addresses, that is, the physical addresses for virtual network registers sets will be different.
- ♦ Restriction of access to virtual sets is done via an MMU function, not by the target.
  - ♦ This sets the minimum size of a region at Nx4K bytes, where N is the number of virtual networks supported by the target.
- e. Because the number of virtual networks supported from release to release can change, care needs to be taken when building up address maps so expansion can take place easily.
- f. A field in the ATP will indicate which of the three regions the request is for.
2. Minimum address size of a target is 256 bytes.
- a. If it is desired to regulate access through the MMU on a per target basis, the minimum address size of a target is 4K bytes and this shall be spaced and enforced outside the target.
  - b. Minimum address size of a region is 4K, so if a target is virtual network aware, it's minimum size is 12K.

## 15.5 Configuration Adapter

1. The configuration Adapter is considered a bidirectional adapter.
2. The configuration adapter will be a unique adapter with unique features.
3. The configuration adapter master interface will be ATP and slave interface will be ATP.
  - a. The interface will support byte enables as allowed in ATP.
  - b. Support of byte enables will be on a register by register basis.
  - c. Customers can separate configuration transactions from other transactions by creating a separate fabric to do so.
  - d. Any Interface that is available as an Initiator ATU can be used to access the configuration fabric.
4. The configuration adapter can have multiple master ATP interfaces as shown in Figure 15-2 on page 236.
  - a. Transactions to different ATP interfaces can be split based on security type, target type, operation type or address range.

**Figure 15-2 Configuration Adapter with Multiple Master ATP Interfaces.**

5. There can be separate configuration Adapters connected to separate configuration networks as shown in Figure 15-3 on page 236.

**Figure 15-3 Separated Configuration Adapters feeding Separated Configuration fabrics.**

6. The multiple interfaces can feed physically separate configuration fabrics or the same configuration fabric.
7. Operation types supported at the target are from 1 to 8 bytes wide
  - a. Operations 4 bytes or less can't span a 4-byte boundary
  - b. Operations greater than 4 bytes can't span an 8-byte boundary

- c. Operation size and starting address must be supported by the register being read or written.
- d. The target ATU on the configuration network that converts the request to the protocol used by a block to program its configuration registers is responsible for breaking up a single request to requests that follow these transaction size rules. The target ATU can limit the max transaction size it supports and its the responsibility of the initiator ATU not to exceed that limit.
- e. The protocol used by the a block to program its registers is considered not a part of this spec and is left to implementation to determine.

## 15.6 Target Standard Interface

- 1. Blocks that contain configuration register will have a standard interface into them to read and write configuration registers.

---

*Multiple protocols can be used by blocks to program their configuration registers, including the ATP protocol. It is the desire that implementation limits the number of protocols used to one, but the inclusion of outside IP can make this desire unenforceable.*

---

## 15.7 Access control

- 1. Access control will be handled by:
  - a. MMU function somewhere on the chip
  - b. Firewalls
  - c. ,ATUs for functions not covered by the Firewall

---

*For instance an Initiator ATU may be configured not to relay access error info back to the external protocol, but the firewall will still be the method used to identify the error and mark the packet in flight.*

---

## 15.8 Reads and writes to non-existent registers

- 1. Targets will complete the transactions.
- 2. Request responses will be mark with an error.
- 3. Reads will return all 0 data.
- 4. Writes will cause no change in configuration space registers except for registers responsible for logging error information.

## 15.9 Configuration Target IDs

1. Every block with configuration registers will start with a unique target ID.
2. Target IDs can be merged for area reduction if it desired for multiple blocks to share a Target ATU.
3. It is allowed to map multiple Target IDs onto a single ATU.

## 15.10 Implementation

This section will be removed once this has been moved into the micro-architecture spec.

### 15.10.1 Configuration Target Interface

The interface into a Target to access its registers will be APB.

### 15.10.2 Target Grouping

In a simple system, the configuration fabric will have an endpoint per target which is composed of a target ATU with an external APB interface. This is a large amount of overhead to cover what could potentially be a single bit register. Because of this Targets can be grouped together to lower the overhead and to take advantage of physical locality of Targets. When this happens, what happens with the target IDs is a function of software. A group of targets could be assigned a single ID or a group of IDs could be given the same route through the fabric.

### 15.10.3 Address Map

When the interconnect has be configured to be virtual network aware, the order of Target addresses in the three regions will be the same. This is done to simplify the task of software when constructing the fabric and configuration address map. Customers should have the ability to insert gaps in the address space to handle the potential of future ECOs and expansions.

### 15.10.4 Minimum Target Address Space

The initial minimum address size of a block with configuration registers will be 4K. Address space will increase as multiples of 4K.

Merging blocks to a single target will not allow the address space to be collapsed.

### 15.10.5 Configuration Register Parameter Definition

Register definitions will be done using the IP-XACT definition.

# 16

# Errors and Interrupts

## 16.1 Interrupts

1. All interrupts are level sensitive.
  - a. 0 indicates no interrupt.
  - b. 1 indicates interrupt asserted.
  - c. An interrupt is generated when an interrupt valid register interrupt bit is 1 and its corresponding interrupt mask register bit is 1.
2. Interrupts can be grouped in any manner and there can be any number of interrupts exiting the NOC.
  - a. Microarchitecture and/or Software can constrain the possibilities.
  - b. This implies you can have any number of levels of hierarchy in your interrupt reporting tree.
  - c. An interrupt can be grouped multiple times.
3. A Single Event will have Single Interrupt Source.

---

### Implementation

Because a Single Interrupt Source can be reported into multiple interrupt groups, it's up to software and implementation to make sure that a single interrupt source is only active in a single group at a time.

---

---

### Implementation

A Single Event can cause other events to occur, which in turn can generate an interrupt. For instance an ECC error in a message can cause the message to be dropped, which in turn will cause a time out error which will generate its own interrupt.

---

4. Multiple Events can be grouped into a Single Interrupt Source.

5. Registers that log multiple events into a Single Interrupt Source must indicate that multiple events have occurred

---

### Implementation

An error Interrupt Data Register that reports ECC errors can store the information associated with the first error of the highest priority error that has occurred since it was last cleared, but must indicate that multiple errors have occurred somewhere in its logging info.

---

6. The block that logs the error information is the block that generates the interrupt.

---

### Implementation

For instance, if an ECC error is detected, and the info is logged locally in the block that detects the error, then that block should generate the interrupt. But, if the error info is transmitted by some means to another block and not stored locally, then the block that receives and stores the info should generate the interrupt.

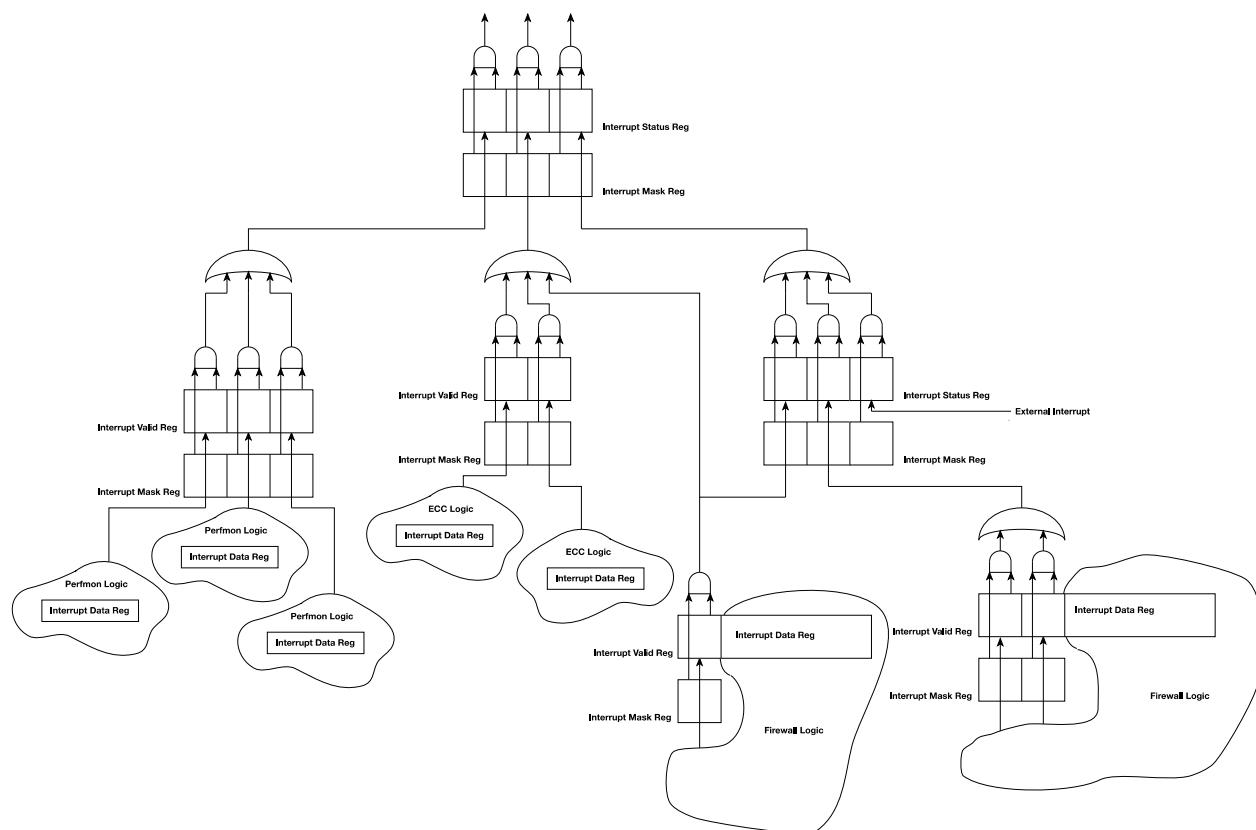
---

7. For request response message pairs, if it is determined locally that the message can continue, the message is marked and allowed to flow until it reaches the initiator ATU, which will log the error information and generate the interrupt.
8. If it is determined locally that the message cannot continue, then it is the responsibility of the block that made the determination to cause an interrupt to be generated.

## 16.2 Interrupt Registers

A typical hierarchy of interrupt registers is shown in Figure 16-1 on page 240.

**Figure 16-1 Example Interrupt Network**



### 16.2.1 Interrupt registers at source.

1. Interrupt Valid register
  - i. Register is mandatory
  - ii. Multiple interrupts can share an interrupt valid register.
  - iii. Interrupt valid bit goes to 1 when state is 0, and input is 1

- iv. Interrupts are cleared by writing 0 to bit corresponding to that interrupt valid bit.
  - v. Status data can sit in interrupt valid register
  - vi. Register is readable and writeable (write 0 to clear an interrupt valid bit).
2. Interrupt Mask register
- a. Register is mandatory.
  - b. One mask bit for every interrupt valid bit.
  - c. 1 indicates interrupt is enabled.
  - d. 0 indicates interrupt is disabled. Default value.
  - e. register is readable and writeable.
3. Interrupt Data register(s)
- a. Register is optional.
  - b. Can be multiple registers.
  - c. Status can be in an Interrupt valid register.
  - d. Register must be readable and maybe writeable.
  - e. Contains data associated with an interrupt.
    - ◆ Examples
    - ◆ ECC interrupt status could contain syndrome, logic/SRAM location, address on SRAM that error occurred on.
    - ◆ ATU error status could contain packet type, packet source, packet destination, error type
  - f. During the configuration of the Interrupt registers, the user will be given a menu of data that is associated with an interrupt from which he can choose for the data that is recorded in the Interrupt Data Register(s).

## 16.2.2 Interrupt Accumulation Registers.

1. Interrupt Status register
- a. Register is mandatory
  - b. Can contain more than one interrupt.
  - c. Input can be an OR of multiple input interrupts of multiple types.
  - d. Contains no status
  - e. Goes to 1 when state is 0 and 1 is on input.
  - f. Goes to 0 when state is 1 and 0 is on input.
  - g. Register is not writable.
2. Interrupt mask register
- a. Register is mandatory.
  - b. One mask bit for every interrupt/valid bit.

- c. 1 indicates interrupt is enabled.
  - d. 0 indicates interrupt is disabled. Default value
  - e. register is readable and writeable.
3. External Interrupts
- a. External interrupts can be brought in as an input into an Interrupt Accumulation Registers.
  - b. External interrupts will come from ATUs and so all necessary transformations (polarity, level vs. edge) will happen in the ATU.

### **16.2.3 Domain Interrupt Masking Register(s)**

- 1. If a power domain or a clock domain has interrupts that exit its boundary there shall exist a Domain Interrupt Masking Register.
- 2. Every interrupt that exits the domain will have a corresponding bit in the Domain Interrupt Masking Register(s).
- 3. There can be multiple Domain Interrupt Masking Registers if the number of interrupts require it.
- 4. 1 indicates that the interrupt must be 0 for the domain to enter sleep ready mode.
- 5. 0 indicates that the interrupt can be any state for the domain to enter sleep ready mode.
- 6. Register(s) is readable and writeable.

## **16.3 Packetization**

- 1. Interrupts can be packetized and sent across the interconnect from a ATU initiator to an ATU target.
- 2. These ATU targets can have other functions other than just interrupts so the packet protocol must differentiate between interrupt packets and other packet types.
- 3. The packet protocol will emulate what an interrupt wire does and that behavior will be reflected on an interrupt wire associated with the ATU generating the interrupt.

## **16.4 Errors**

- 1. Errors are considered a subset of the Interrupt definition.
- 2. To create an error that is polled as opposed to using an interrupt, the mask bit is set to 0.
- 3. If the amount of data associate with an error is large, instead of being placed in Interrupt Data Registers the mechanism used the debug/trace mechanism that is architected in section TBD.

4. For cases where only a single set of status covers multiple errors, the following rules will be followed:

- a. Some status will indicate if multiple errors have occurred.
- b. Only the error status of the error with the highest priority will be reported.
- c. If two errors of the same priority have occurred, the error status for the first error to reach the status register will be reported.

---

#### Implementation

---

Take for example an error status register associated with a group of SRAMs that report ECC errors. Double Bit Errors (DBEs) will have higher priority than Single Bit Errors (SBEs), so if an SBE occurs before a DBE, the SBE error status will be overwritten by the DBE error status and a bit will be set indicating multiple errors have occurred. If the opposite happens, the DBE status will not be overwritten, but a bit will be set saying multiple errors have occurred. If two DBEs occur, the error status of the first error will be reported and a bit will be set that indicates multiple errors have occurred.

---

5. Errors associated with messages shall have the ability to have the following information logged

- a. Error Type
- b. Error Syndrome
- c. Source ID
- d. Target ID
- e. Operation Type
- f. Address if available
- g. Block ID where error detected (Same as source or target ID if they exist for block)
- h. Customer configurable metadata

6. Errors associated with blocks shall have the ability to have the following information logged

- a. Error Type
- b. Error Syndrome
- c. Operation Type if associated with an external protocol transaction
- d. Address if available
- e. Block ID where error detected (Same as source or target ID if they exist for block)
- f. Customer configurable metadata

## 16.5 Implementation

1. To be removed upon completion of the MicroArchitecture Spec or Software Architecture Spec.

## **16.5.1Initial Interrupt Hierarchy (Software starting point)**

### **16.5.1.1ATUs**

1. An ATU will have a set of Interrupt Accumulation Registers that accumulates all internal interrupts into a single interrupt.

### **16.5.1.2Fabrics**

1. A fabric will have a set of Interrupt Accumulation Registers that accumulates all internal interrupts into a single interrupt.

### **16.5.1.3Domains**

1. A domain will have a set of interrupt accumulations registers that accumulates all into a single interrupt and that single interrupt will feed the Domain Interrupt Masking registers.

### **16.5.1.4Hierarchy**

1. Where ever a group of blocks, fabrics, domains are grouped into a single RTL block, there will be a set of Interrupt Accumulation Registers that accumulate all the internal interrupts into a single interrupt. This includes the “Top Level” block.

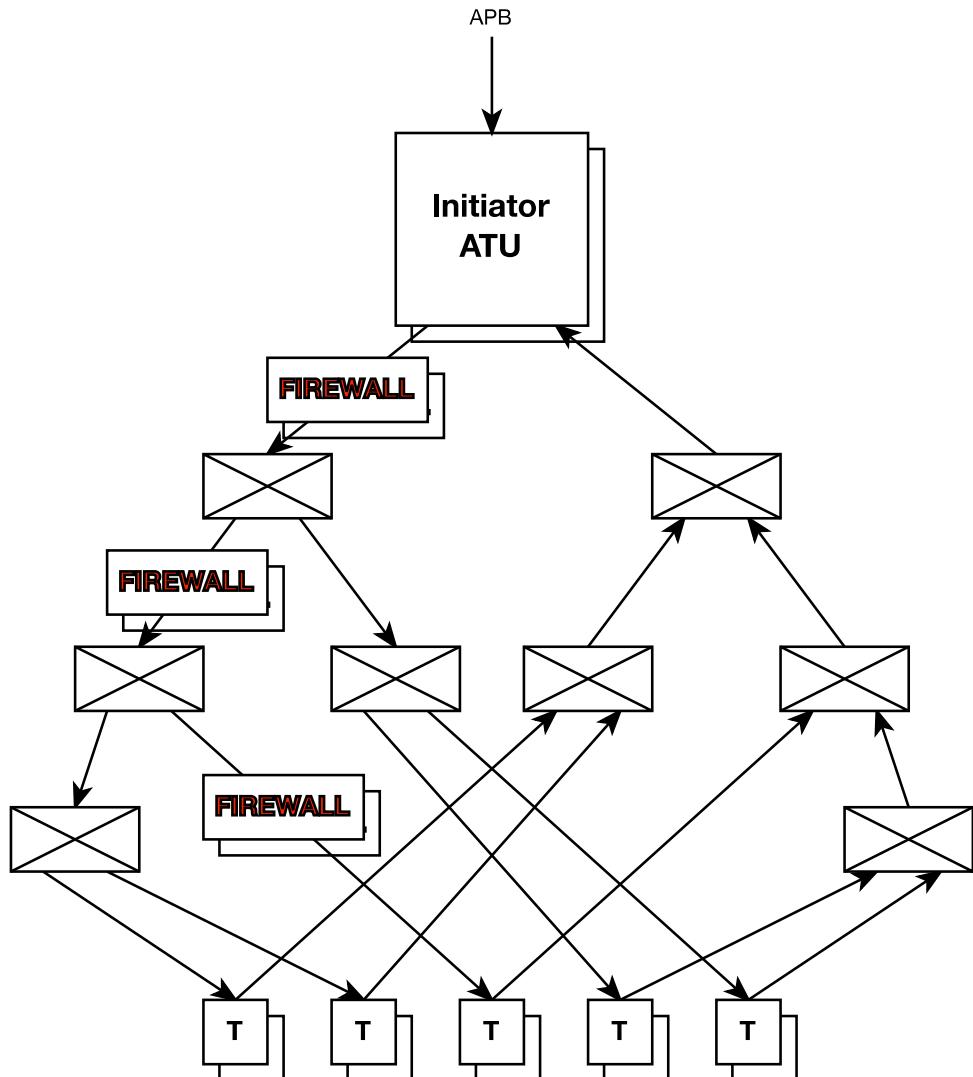
# Resiliency

Resiliency are the methods provided to increase the reliability and Failure In Time (FIT) rates and report errors detected in a manner that allows fault tolerant products to be built using Symphony.

The use of Shall be and will be in this chapter is in the context of providing a particular function and assumes that all functions are desired. The way to read it is “If you want this function you shall do this.” Resiliency is in itself optional, as is its various pieces.

The block diagram shown in Figure 17-1 on page 245 gives examples of how the various protection schemes described below will be used.

**Figure 17-1 A Fabric with Resiliency**



## 17.1 Data Modifier Blocks

The word data is used generically and refers to both header and payload.

1. A data modifier block is any block of the interconnect that can modify data, which can be the contents of a packet.
2. Data modifier blocks will be duplicated to provide fault detection.

## 17.2 Pass Through Blocks

The word data is used generically and refers to both header and payload.

1. A pass-through block is any block that always passes data through unmodified.
2. Data passing through a pass-through block will be protected by parity, Triple Modular Redundancy (TMR), ECC, CRC, or any combination of the four.
3. If the pass-through block uses data from the packet to perform its function, when desired, a method for using corrected data shall be provided should the protection scheme used provide an option to correct data (such as ECC).

## 17.3 Control Blocks

The word data is used generically and refers to both header and payload.

1. A control block is any block that provides only control to the data path.
2. Control blocks can be protected with ECC or duplication.

## 17.4 Data Path Blocks

The word data is used generically and refers to both header and payload.

1. A data path block is a block that operates on data of a message.
2. Data path blocks can be pass through blocks or data modifier blocks, which determines their method of protection

## 17.5 Memories and Registers

Memories and registers will be referred to generically as memory.

The word data is used generically and refers to both header and payload.

1. A memory is any storage element that uses a bit cell to store data.

2. The data in a Memory will be protected by parity, TMR, ECC, CRC, or any combination of the four.
3. The address to a Memory will be protected by parity, TMR, ECC CRC or any combination of the four.
4. Memories inside a packet modifier block shall not be duplicated. Inputs and outputs from the packet modifier block to the memory will be treated like other inputs and outputs and outputs of the packet modifier block to the memory will be included in the compare circuitry.

## 17.6 Block Composition

1. Blocks can be decomposed into sub-blocks.
2. Sub-blocks can then be protected independently of other sub-blocks in the block.

---

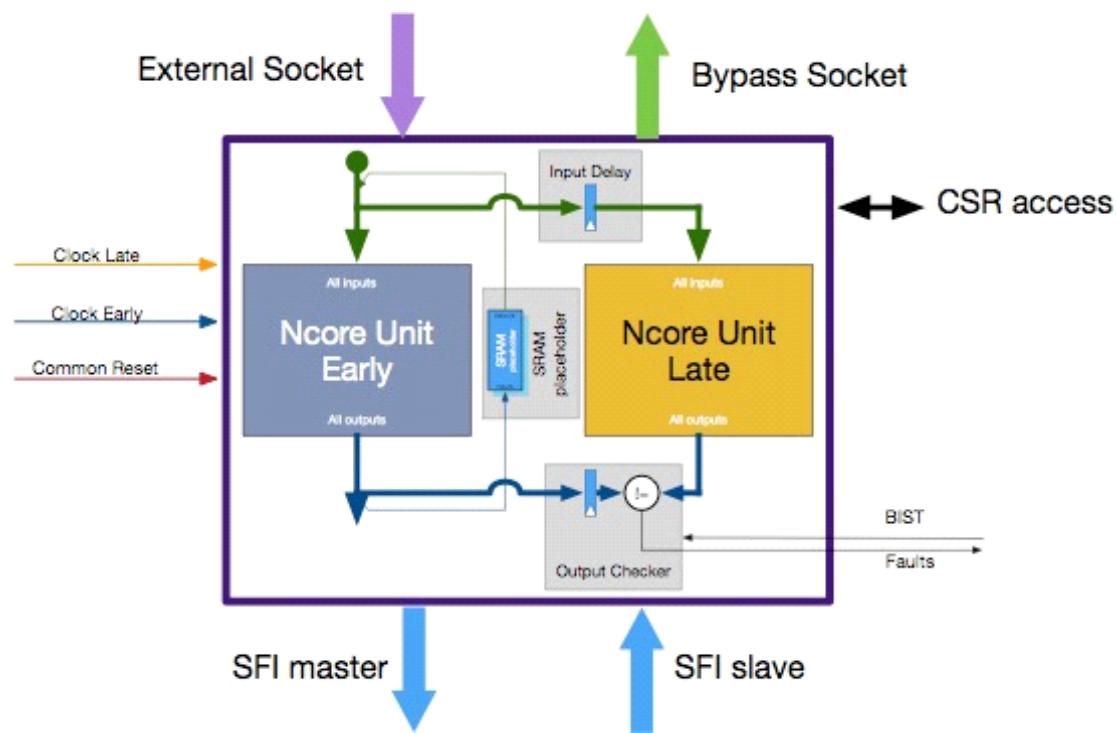
### Implementation

Imagine an ATU. According to the above, the ATU would be a data modifier block, so the designated protection scheme would be duplication. However, an ATU can contain Memory. Since Memories are large and can be protected with ECC, the ATU can be broken apart into two sub blocks; the memory and the rest of logic. ECC can be used on the memory, and duplication used on the rest of the logic. The rest of the logic could be further broken down into control and data path blocks, and the data path blocks could be further broken down into path through blocks and data modifier blocks.

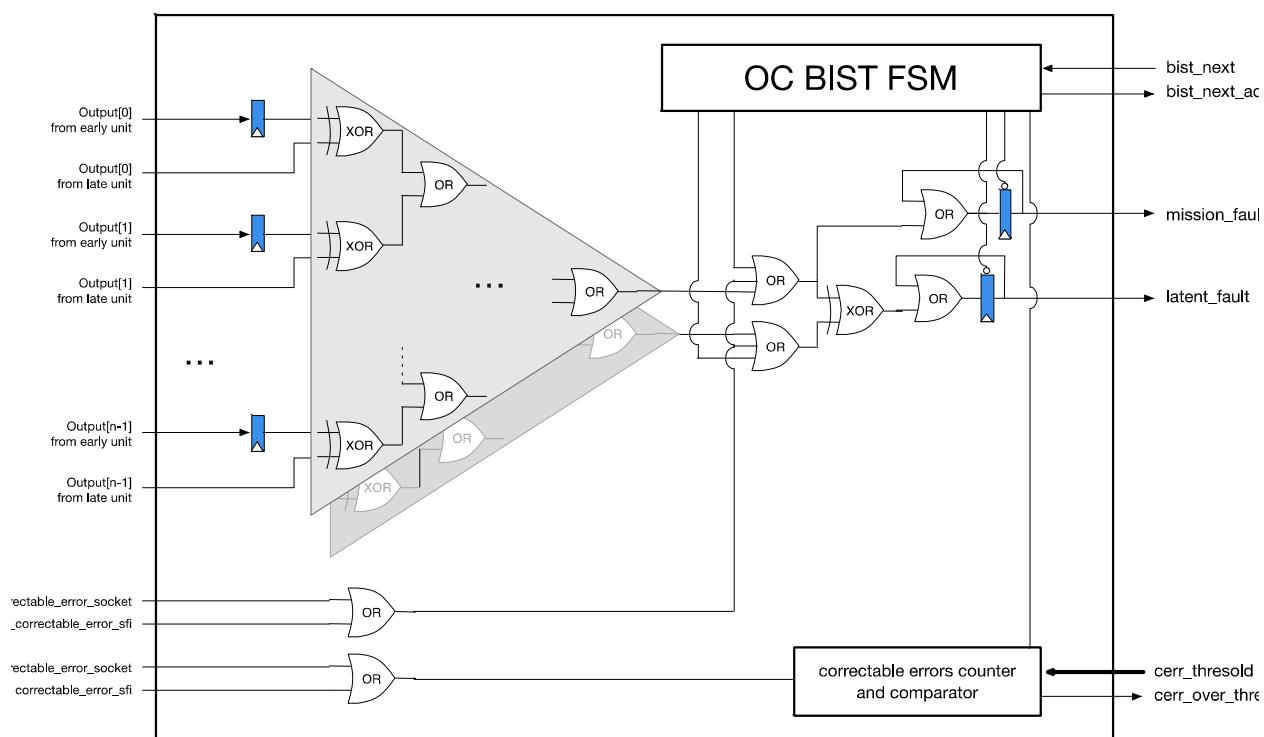
---

## 17.7 Block Duplication

A block diagram of how a block is duplicated is shown in Figure 17-2 on page 248.

**Figure 17-2 Block Duplication Diagram.**

1. One of the blocks will be designated as the master and will drive the outputs to the rest of the interconnect.
2. One of the blocks will be designated as the slave and shall run from 0 to N clocks delayed from the master in half clock increments.
3. Physically separate clocks drive each block and this physical separation extends past the boundary of the interconnect.
4. *The properties of the separated clocks outside the interconnect affect the FIT rate and so assumptions must be documented and communicated to the customer.* A compare circuit will compare the outputs of the Master and Slave blocks and generate a signal when the outputs don't compare.
  - a. The compare circuit shall be duplicated along with the generated signals.
  - b. The outputs from both blocks will be compared and to generate “latent fault” signals.
  - c. The outputs from one of the compare circuits shall be designated as the “mission fault” signals.
  - d. All the latent faults will be ORed together to generate a latent fault Error.
  - e. All the mission faults will be ORed together to generate a mission fault Error.
  - f. All inputs into the latent fault OR tree will be XORed with a single signal.
  - g. All inputs into the mission fault OR tree will be XORed with a single signal.
  - h. To scrub faults, the latent fault Error and mission fault Error signals will be masked and then various combinations of the two signals described in f and g above and the two Error signals will be compared against expected results. The block diagram is shown in Figure 17-3 on page 249.

**Figure 17-3 Output Checker Block Diagram.**

## 17.8 Packet Protection

1. A packet will be protected as it passes through pass through blocks with parity, TMR, ECC, CRC codes, or any combination of the four.
  - a. A bit in a packet can be covered by multiple protect schemes.
  - b. When protected, all bits in a packet will be covered by one of the stated means.
  - c. For resiliency concerns, side band signals on a link in the NOC are considered part of the packet unless it can be shown that the failure of the bit will not affect proper function, but could affect performance.
2. When a packet is modified, for what ever reason, the packet protection will be regenerated.
  - a. For protections schemes that can correct data, corrected data will be used when regenerating protection.
3. When a packet is modified, for what ever reason, the protection will be decoded and any errors detected will be logged in some manner.

## **17.9 Error detection**

1. Since packet modifier blocks are fail fast pairs, they will detect all single faults that occur and a majority of multiple faults that occur.
2. TMR associated with Packets will detect all single faults that occur for the section associated with TMR.
3. Parity associated with packets and Memories will detect all single faults that occur for the section covered by parity.
4. CRC associated with packets and Memories will detect all single faults and the majority of multiple faults that occur for the section covered by CRC.
5. ECC associated with packets and Memories will allow the memories and packets to function in the presence of a single fault, detect all double faults that would cause a functional fail for the section covered by ECC.

## **17.10 Fault Tolerance**

1. Fault Tolerance is never guaranteed in a packet modifier block.
2. Fault Tolerance is guaranteed in the presence of a single fault on memory storage elements and in the data path between initiator and target ATUs when the logic elements use corrected data from the packet to perform their function along the path.

## **17.11 Safety Controller.**

1. The Safety Controller accumulates all the error signals from the various blocks and generates a single set of signals to the outside world.
  - a. latent fault interrupt
  - b. mission fault interrupt - Includes uncorrectable errors from SECDED ECC codes.
  - c. correctable error over threshold interrupt
2. The Safety Controller will provide a means to be controlled and have its state queried over the configuration network by means of a standard register interface.

---

*Architecturally we can control access to these registers by three means: Firewalls, Physically separate configuration network or MMU enforced access control. Please read configuration chapter for more details.*

---

3. The Safety Controller will provide a means for scrubbing it's latent faults and the latents faults in the Output Checker BlockCustomer Embedded Protection
1. A method shall be provided to enable customer code than can encode or decode their own custom protection that they have implemented on top of an external protocol

2. The customer's implementation must not pipeline from data in to data out of block (data can be header/command/data from external protocol)
3. The customer's implementation must provide the following methods:
  - a. Decoding from a request
  - b. Encoding of a request
  - c. Decoding from a response
  - d. Encode of a response
4. The customer's decoder will provide a single output called "fail."
  - a. The signals needs to only assert for one cycle that it detects an error.
  - b. The support code surrounding the customer's code will turn this into an interrupt that needs to be cleared.
  - c. If the customers wants to log information associated with this error the logging must be implemented in their registers space. (doesn't follow the Symphony interrupt standard register implementation.)
5. Customer's code will be given access to inputs that can drive from the boundary of the top macro.
6. Customer's code can drive outputs that go to the boundary of the top macro.
7. Customer's code can contain registers that the customer will be given access to over the configuration network and they implement using a stand APB interface.

---

#### Implementation

The above does not imply that protection schemes that can't correct and span multiple beats, but any such method wouldn't be able to correct any data unless the error falls in the last beat.

---

---

#### Implementation

Whether the inputs and outputs of the customer code can go to places other than I/O at the boundary of the top macro will be a function of software. It can be envisioned that these I/Os could be used to drive into the debug and trace logic or interrupt trees of the interconnect.

---

## 17.12 Fail Operational

1. The RTL output from the interconnect generation tool must enable the end customer to physically separate duplicate logic masters from slaves and from the rest of the interconnect logic.

---

*This is to enable the insertion of BIST logic by the customer which can be used to determine when there is a fault generated by duplicate logic which block is faulty, the master or the slave.*

---

2. Implementation must enable the use of the duplicate logic slave as the master if the duplicate master is discovered to be faulty.
3. When a duplicate master or slave is determined to be faulty, implementation must enable the masking of duplicate errors from that set of duplicate logic.

## 17.13 Request and Response Timeouts

1. There will be timers in both the target and initiator ATUs that track packets, requests and responses. Refer to the ATU specification for details.

## 17.14 Protocol Violations

1. If there is a protocol violation detected inside the fabric, the packet will be dropped and an interrupt generated.

## 17.15 Parity of ECC error detections

1. If there are errors detected that prevent the packet from being properly routed, then the packet will be dropped and an interrupt generated.

## 17.16 Spurious Packets

1. If a spurious packet is detected, like a response that has no request or a request that lands in the wrong requestor, then the packet is dropped and an interrupt is generated. Similar violations in external protocols are covered in the ATU chapter.

## 17.17 End to End Protection

1. Data into and out of the interconnect must have some protection scheme.
2. When data is transferred from one protection domain to another, the two domains must overlap.

---

### Implementation

For example, there is a request port where all signals of the interface to the external protocol are covered by ECC into the Initiator ATU. At the other end there is a Target ATU where all the signals of the external protocol interface are covered by ECC but it's a different external protocol than the external protocol associated with the Initiator ATU. In this case a translation must occur.

The translation could be conceivably done at a register where the incoming protection scheme is clocked into a register and checked on the register's output while at the same time the register's corrected output data is being used to generate the outgoing scheme's protection bits. One problem with this is there is always going to be some wires where errors can be seen in one protection domain but not the other, so the new protection domain will have a finite possibility of encoding bad data while the old protection scheme sees no error. The other problem is there may be no register available to use. Because of this, and the nature of synthesis, it can't be guaranteed overlap unless the code manually implements the logic and it is don't touched from synthesis. Not a good thing to do in IP where the target standard cell library is not known.

To solve both of these problems the plan is wherever one of these conversions occurs, it is a Data Mod-

fier Block and the conversion is done in a duplicated pair. This fulfills the intent of 17.1 rule 2 and 17.17 rule 2.

---

## 17.18 Implementation

### 17.18.1 Packet Internal Protections

In the initial implementation, the protection scheme will not extend past a PHIT.

Protection can be sub-PHIT and bits can be double covered and the schemes do not need to match.

The idea behind this is it may be desirable to put ECC protection on the bits in the header that are used to route a packet in a switch so that it covers less bits making the logic faster and enabling the use of corrected data by the switch. When this is done, the routing bits maybe double covered. For instance, the entire header maybe covered by parity or ECC and the routing bits are covered by a separate ECC code.

### 17.18.2 Phase Shift of Duplicated Logic

Whole ratio phase shifts will only be supported in rev1.

The reality is a latch in a chip that supports scan isn't much smaller than a flipflop and somewhere along the delay path an additional half cycle will need to be added to align the signals back to the reference clock, so there's no advantage of reporting an error a half cycle earlier.



# 18

# Power and Clock Management

This chapter describes the proposed power management solution for Symphony and NCORE. The chapter starts with defining terminology that will be used later in the document. The chapter also describes at a system level how the proposed solution will work and fit in the overall scheme of having multiple power modes and domains in the SoC.

It should be noted that Symphony/NCORE fabrics do not manage their own Clock and Power domains. This will be done by an outside agent that will be referred to as the Power Management Unit (PMU).

This Chapter also describes Symphony's power and clock management scheme. This includes power gating, different levels of clock gating and dynamic voltage and frequency scaling. However, the chapter does not describe the circuitry and synchronizers that sit at the boundary of clock and power Domains or Regions. Those details are covered in the micro-architecture specification.

## 18.1 Regions vs. Domains and Sub Domains

A clock or power region is a set of logic driven by one clock or one power supply. A domain is a set of logic that is controlled, conceptually, by a single set of Domain Interface Signals. A single region can have one or multiple domains in it.

Sub domains only exist for clocks. A clock domain has one or more sub domains in it. A sub domain is a set of logic that is clocked by a single clock that has a known synchronous relationship to all the other clocks driving all the other sub domains in a clock domain.

## 18.2 Power & Clock Domains

Symphony/NCORE can be partitioned into multiple power and clock domains. The system designer can partition the network as they seem fit as long as the following rules are observed.

### 18.2.1 Domain Spanning rules

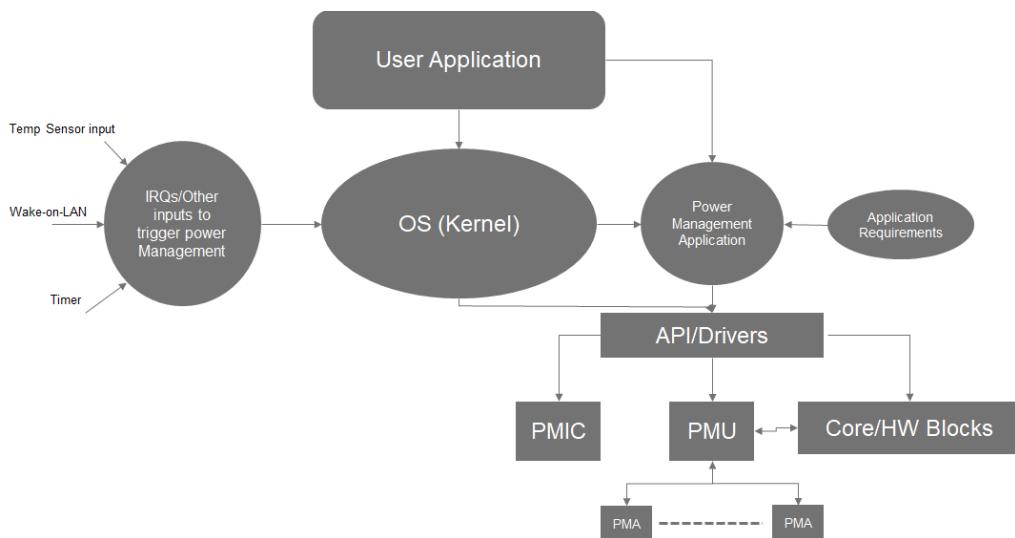
- ▶ No clock domain will span multiple power domains.
- ▶ A power domain can span multiple clock domains.

## 18.3 Power Management System View

Developing a cohesive strategy for power management in an SoC is a serious and tedious endeavor since it impacts all aspects of the device ranging from hardware capabilities to what can be done in software and the capabilities of the power management application. For starters, the system designer has to: 1) understand the capabilities of the hardware blocks and what power/clock modes the hardware blocks are capable of transitioning into; 2) the dependencies hardware blocks may have with respect to other hardware blocks in the system (more on this later in the document); 3) the capabilities of the power management application and associated drivers; and finally 4) the quality of service requirements of the application. Each of the above will be discussed here in some detail, while keeping the discussion at the system level.

A typical system would consist of the user application running on some Kernel, or hypervisor or bare-metal. The system will also have a set of defined events that will serve as triggers to shift the system from one power/clock state into another power/clock state. For example, the SoC may transition in to a lower power state if the on-chip temp sensor exceeds a certain pre-configured temperature range. The sensor will generate an interrupt that will be captured by the Kernel. The Kernel may start the power transition routine or call the Power Management application to manage the transition. The Power management application interfaces with the HW blocks on the SoC, the PMU (Power Management Unit) and the PMIC (Power Management IC) as shown in the Figure 18-1 on page 256 below. and sends them the appropriate commands to reach the desired power/clock state.

**Figure 18-1 Overall system view of how the SW would interface with HW elements**



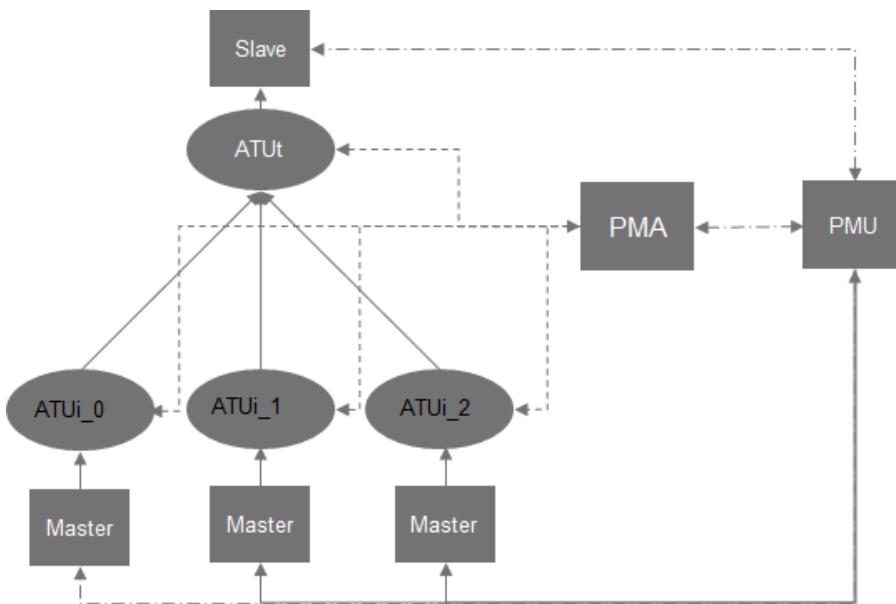
Each hardware block in the SoC that participates in power modes of the SoC, has to register its capabilities with the power management application. These capabilities will include the different voltages, frequencies and the power modes (sleep, deep sleep, on and power/clock off) the HW block supports.

The application QoS requirements also dictate which clock/power modes can be invoked on the SoC. The QoS requirements of the application with respect to power management will determine whether the transition time needed to enter and leave a power/clock mode and the residency time in the state is acceptable to the application. The QoS requirement may also state that the application is an always-on application in which case the power management application would never put the device to sleep.

Knowing the properties of a HW block is not sufficient in determining whether the block can be transitioned out of the current state. The power management application in addition to the properties of the hardware block, needs to determine the dependencies the block may have on other blocks in the SoC. These dependencies need to be addressed before the HW block can transition out of the current state.

These dependencies are captured in what we refer to as the “*Power Dependency Tree (PDT)*”. The root node in the PDT is the HW block that has to go through the power/clock state transition. The other nodes in the tree including the intermediate and the leaf nodes represent the dependency that has to be

removed or satisfied before the root node can transition. Consider the PDT in Figure 18-2 on page 257 below:



**Figure 18-2 PDT for turning off the Slave**

If the system wants to transition the Slave in to a different power management state (e.g., to Sleep) from an active state, then before the slave can transition the power management application has to follow the above PDT. As shown in Figure 18-2 on page 257, starting from the leaf nodes, the power management application will quiesce the three Master nodes. Quiescing the Master nodes would mean that no transactions are sent from the Master nodes to the slave and all pending transactions at the Master nodes are complete. Once that is accomplished, the power management application will transition the ATUi's in to the quiescent mode. The power management application will work its way up the tree and quiescent the ATUt and finally power off the Slave.

The power management application will tree from the root to the leaf nodes when turning on the Slave and

In the section below we formally define the PDT.

### 18.3.1 Power Dependency Tree (PDT)

PDT defines the relationship between different entities and/or actions in the SoC that should be respected before a particular HW block or a power/clock domain changes power/clock state.

PDT depends on how a particular application uses the SoC and hence is very application dependent.

The PDT does the following:

- ▶ Establishes a parent-child relationship between entities that are dynamically power/clock managed. The PDT includes not only the relationship between HW blocks but also flows, interrupts etc.
- ▶ Establishes conditions before clock/power state change can be initiated

The application walks the tree to make sure the Child node is in the “appropriate” state before changing the state on the current/parent node

PDT is not only limited to HW blocks but can also include/used for power domains in the system. Power Management applications are generally aware of these relationships. For example, if a processor core has to be put to sleep or in power off mode, then:

- ▶ A barrier is setup and the core stops issuing any new transactions.
- ▶ The power management application ensures that all outstanding transactions have completed.
- ▶ The cache is flushed.
- ▶ All interrupts to the core are masked except for the interrupt generated from events that will wake the processor core.
- ▶ If the processor core is working in SMP mode, then it is removed from that mode and put in the AMP mode so that other processor cores stop sending transactions to this core.
- ▶ Once of the above are satisfied, then and only then the processor core is put in to the sleep or power off mode.

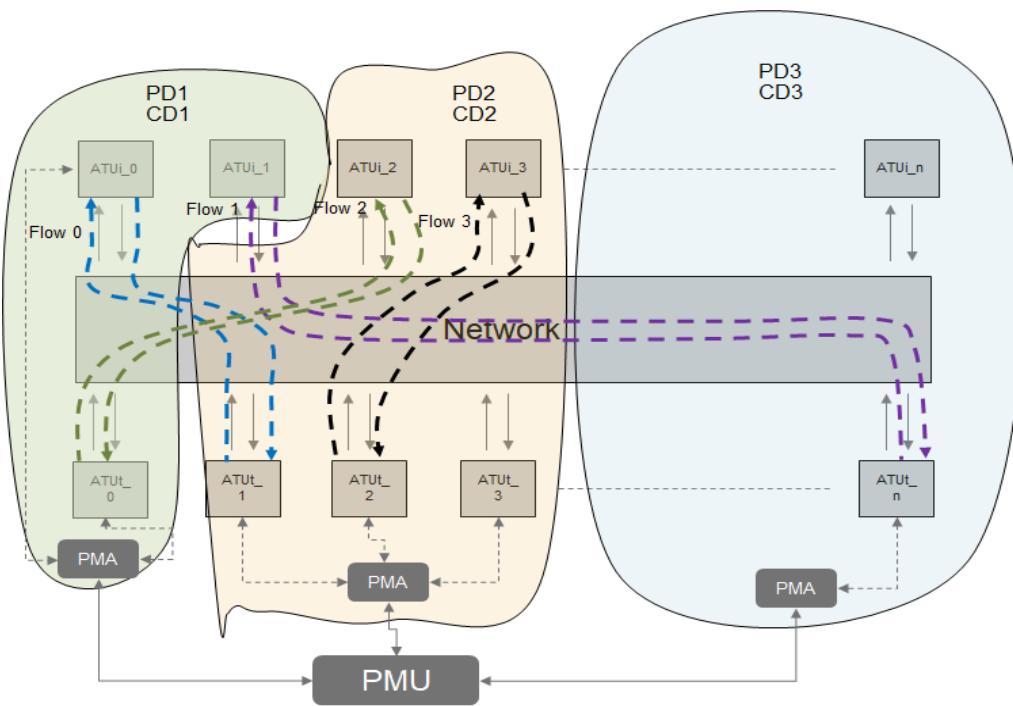
All of these steps would be defined in PDT and/or captured by the power management application.

## 18.4 PDT and Maestro

In an SoC, Socket level communication is carried over the NoC. The NoC topology could be a simple bus, a regular network topology such as a Mesh, or a non-regular topology such as a tree. Flows corresponding to Socket Level communication are routed over the NoC and may take several hops to reach the destination HW block. Flows could be routed either automatically by Maestro in the case of regular topologies or manually by the System Architect. In either case, flows can traverse multiple network elements and power domains. Since flows may traverse a power domain that they do not either originate or terminate, the PDT has to be aware of the existence of this flow and the power management application would need to quiesce this flow before it can turn the power domain off. The following example, illustrates the need for the PDT to be cognizant of all the network elements in the NoC, their dependencies and the route of all the flows in the NoC.

Consider the system shown in the Figure 18-3 on page 259 below:

**Figure 18-3 SoC with multiple power domains and flows crisscrossing the power domains**

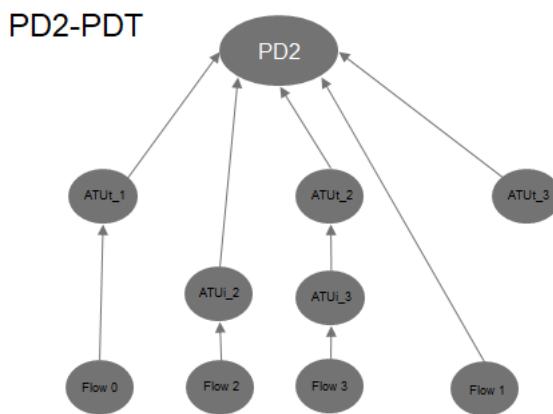


In Figure 18-3 on page 259, the SoC is partitioned into multiple power domains. Each power domain has within its domain several network elements including the Initiator ATUs (ATUi), the target ATUs (ATUt), width adapters and switches. Each power/clock domain is controlled by a Power Management Agent (PMA). The PMA will be discussed later in the chapter. Each PMA in turn is connected to the Power Management Unit (PMU).

The PMU is usually a small processor that runs the power management application.

Flows between the Initiators and the Targets are mapped on to the NoC. As can be seen in Figure 18-3 on page 259, Flow 0 between ATUi\_0 and ATUt\_1 crosses two power domains, namely PD1 and PD2. Similarly, Flow 1 between ATUi\_1 and ATUt\_n crosses three power domains, namely PD1, PD2, and PD3. Other flows such as Flow 2 and Flow 3 between ATUi\_2 and ATUt\_0 and ATUi\_3 and ATUt\_2 respectively, are routed on the NoC and cross multiple power domains as well.

Since the SoC is partitioned in to 3 separate power domains, each of these power domains can be turned off independently. Suppose PD2 was to be turned off. The PDT for powering off PD2 is shown in Figure 18-4 on page 260' below:

**Figure 18-4 PDT for turning off PD2**

Notice that the PDT starts at the flow level. It also includes Flow 1 which neither originates from nor terminates in PD2. However, Flow 1 is routed through the network elements in PD2. As such Flow 1 has to be quiesced before PD2 can be turned off.

The power management application will walk through the PDT for PD2. It will first quiesce flows 0, 1, 2, and 3. It will then query ATUi 1, 2, and 3 to inquire if they could be turned off. The PMA will inform the PMU whether the elements are ready to be turned off or not. Once the ATUi's and ATUt's are turned off, the power management application reaches the root node and turns PD2 off. Notice, that only Flow 1 is quiesced. ATUi\_1 and ATUt\_n need not be turned off for PD2 to be powered off.

Maestro has knowledge of the routes that flows take and can pair flow information with power domains they cross. Maestro also has knowledge of how the flows are mapped to different HW blocks within the NoC/SoC. In the above example, Maestro has the information about which flows traverse PD2 and which network elements reside in PD2. It also has information about which flows originate and terminate in PD2. Maestro can provide this information to the power management application for the application to build out the PDT. The power management application can query Maestro about the following via appropriate APIs provided by Maestro:

- ▶ What network elements including end points are in the power domain in question.
- ▶ What flows (Src-Dest pair) traverse which power domains and network elements.
- ▶ How are the flows mapped on to different network elements. That is, the application can query the complete route including network elements that a socket connection traverses.

It is important to note that while the System Architect is aware of which end points lie in which power domain, they would not have complete information on which network elements such as switches, width adapters, multicast stations, etc. reside in which power domain, or how flows are routed if the topology and mapping of flows is done automatically by Maestro. As such Maestro has to be able to provide this information to the power management application.

## 18.5 PDT and Power Adapters

Power Adapters are bi-directional adapters and are inserted at the boundaries of the power domain. The request flows and the corresponding response flows are mapped to traverse the same power adapter. The power adapter records the number of request transactions and corresponding response transactions. If there are outstanding transactions at the power adapter, the adapter would inform the PMA that it is busy and the domain cannot be powered off. When there are no outstanding transactions at the power adapter, the adapter will signal the PMA that it could be powered off. If a packet was sent to

a power domain that was turned off, the power adapter would return the packet back to the Initiator with an error.

The power adapter imposes the restriction that the responses have to pass through the same power adapters as the requests. This restriction imposes additional burden on the routing algorithm to make sure that this condition is satisfied all the while keeping the network deadlock free.

The power adapters are also not sufficient for implementing an orderly shut down of a power domain or a hardware block. If appropriate steps, as described earlier, are not taken, packets may continue to be sent to the powered down domain resulting in errors sent back to the Initiator.

In Symphony, the initiator ATU will raise an interrupt if it receives a packet with transport error. The only way to orderly and safely shut down a power domain is to construct a PDT as explained earlier. The PDT will make sure that all necessary steps are taken so that no transaction is sent to the domain that is requesting to be powered off and all the elements in the domain are quiesced. In fact, if the PDT is constructed properly, Power adapters are not needed inside the network.

The PDT is sufficient and necessary to properly put a power domain or a HW block in different power/clock modes. PMA's assist the power manager application in determining when a particular HW block can be turned off, as the application steps through the PDT.

Maestro can support insertion of power adapters at the ATUis and ATUs for R1 release of Symphony. Insertion of power adapters before the ATUis and ATUs will provide the user with additional security to ward against any race conditions that may occur with respect to turning off the power domain immediately following the ATUs.

Power adapters can still be used to enhance system debug capability. Alternatively or in conjunction with power adapters, the timeout mechanism in the Initiator ATUs could also be used to assist with debugging the overall system.

Operating system such as Linux already support the concept of dependency tree. For example the `runtime_idle()` API is a Linux Kernel Power API which can only be executed by the PM application if the following conditions are met

- ▶ Current device is idle: `usage_counter == 0`
- ▶ All children in the dependency tree are idle. `active_children_count == 0`

Linux genpd API for managing Power domains supports the following capabilities:

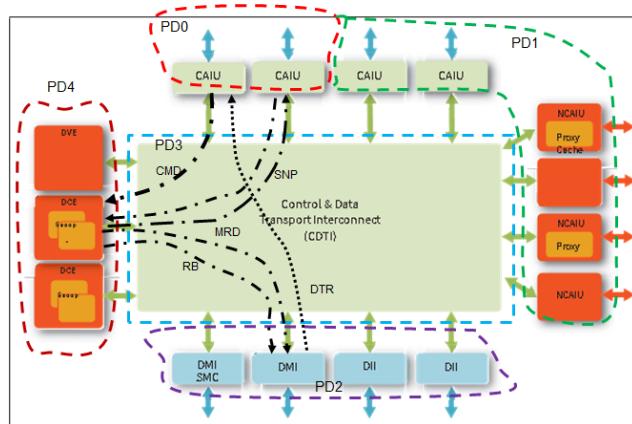
- ▶ Power Hierarchies
- ▶ Adding and removing HW blocks from domain in runtime.
- ▶ Power governors. Power governors define power policies for the domain, such as an always-on policy.

## 18.6 PDT and NCORE

Unlike Symphony, in NCORE a single transaction can spawn off multiple transactions to different HW blocks in the SoC. These transactions may be routed over different Fabrics or networks. Since the power adapters have only local knowledge, they are inadequate in preventing another non-adjacent power domain from going to sleep or power off mode and can result in a transaction never reaching its destination.

Consider the NCORE network shown in the Figure 18-5 on page 261 below:

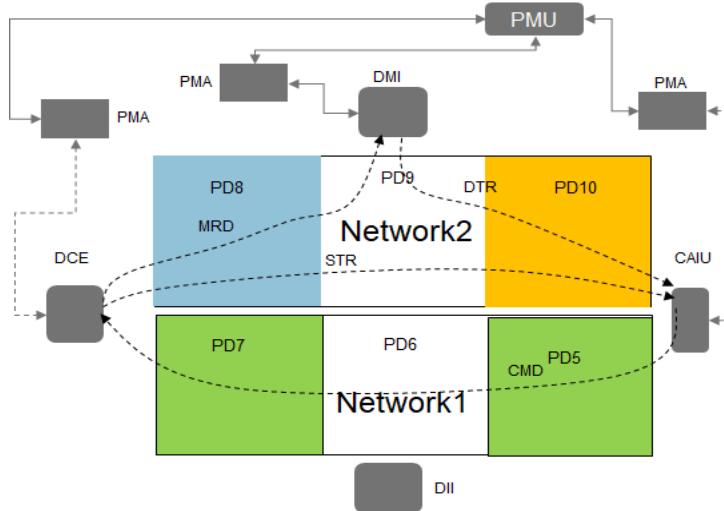
**Figure 18-5 : Sample NCORE network and Flows**



Multiple CAIUs talk to a single DCE, which in turn communicates with DMI and other CAIUs in the system. The different flows may be traversing not only different networks but also different power domains. The only way to safely put the power domain in off or sleep state is to build a PDT which captures all the dependencies and steps needed to put the domain to sleep.

In Figure 18-6 on page 262 , a sample NCORE system is illustrated. The PMA connects with the NCORE blocks. Each domain will have its own PMA. The PMAs in turn connect with the PMU. If the system wanted to power down the DCE, then a PDT for the DCE would have to be created.

**Figure 18-6 Sample NCORE system with PMAs and PMU connectivity**



This PDT will start at the leaf nodes by quiescing the flows originating at the CAIU and destined for DCE, the flow from the DCE to the DMII and the CAIU and finally the flow from DMII to CAIU. Once all the flows are quiescent, no transaction are outstanding and no transactions are originated to be sent to the DCE. the PMU will then query the PMA connected to the DCE whether it could be turned off. The PMA will in turn query the DCE whether it is ready to go to sleep/power off. If the answer is in the affirmative, the PMU will power the DCE down otherwise it will stay on until it the DCE is ready to go to sleep/power off.

In the next sections we describe the PMA, Q/P-channel signals and sequencing as well as anticipated future work.

## 18.7 Configuration Network

The configuration network could be setup in two ways: 1) use the regular network to send messages from an AXI or APB ATUi to an APB ATUt which then connects with the connects with the network elements that need to be configured; or 2) the other way is to set up a completely separate configuration network. In the former case, in order to shut down the network, a PDT would have to be setup by the customer. Maestro will provide the necessary information to setup the PDT.

In the later case, the configuration PDT would have to setup in much the similar way as for the regular network. If the configuration network is in the “always-on” domain, then the system will rely on coordination between the PMA and PMU and the config processor to make sure no configuration commands are sent to a network element that is currently in the power/clock off state. If such, a command is sent to a network element that is turned off, then the system will have to rely on the ATUi timeout to determine if there was an error or the command did not complete.

## 18.8 Symphony and NCORE Solution Summary:

As was explained earlier in this chapter, the PDT provides sufficient information for the customer to change the power state of a particular network element or a power domain in the SoC. It is the customer’s responsibility to develop the PDT. However, Maestro will provide all the NoC related information required by the customer to build such a dependency tree.

Maestro has complete knowledge of the following:

- ▶ Number of Power Domains in the NoC
- ▶ Network elements per power domain and power states supported
- ▶ Flows that originate/terminate and crisscross which power domains

Maestro will provide this information in the following ways:

1. Export text file with the following information:
  - a. List of power domains and sub-domains.
    - ◆ API : listPowerDomain: provides a list of all power domains in the NOC
  - b. Network elements within a power domain.
    - ◆ API: listNE(<namePowerDomain>): provides a list of all the network elements that reside in the named Power Domain
  - c. Flows that originate, terminate and crisscross the power domain. Information about source and destination of the flows is provided in the text file.
    - ◆ API: listFlows(<namePowerDomain>): provides a list of flows that terminate, originate or crisscross the power domain.
2. Embed the above information in the UPF file format
  - a. The UPF 3.0 file format allows for power domains to be defined. The power domain definition includes “On”, “Off” and “Retention” states of different elements in the power domain.
  - b. UPF 3.0 file format also allows for power domains to be hierarchically defined.
  - c. Current limitation of the UPF file is that flow level description is not possible. However, because dependencies can be defined, the UPF file defining power down state can create a dependency on an element not in that power domain.

3. TCL Commands to export the above information to file <filename>:

- listPowerDomains <filename>
- listNE(<name\_powerDomain>) <filename>
- listFlows(<name\_powerDomain>) <filename>

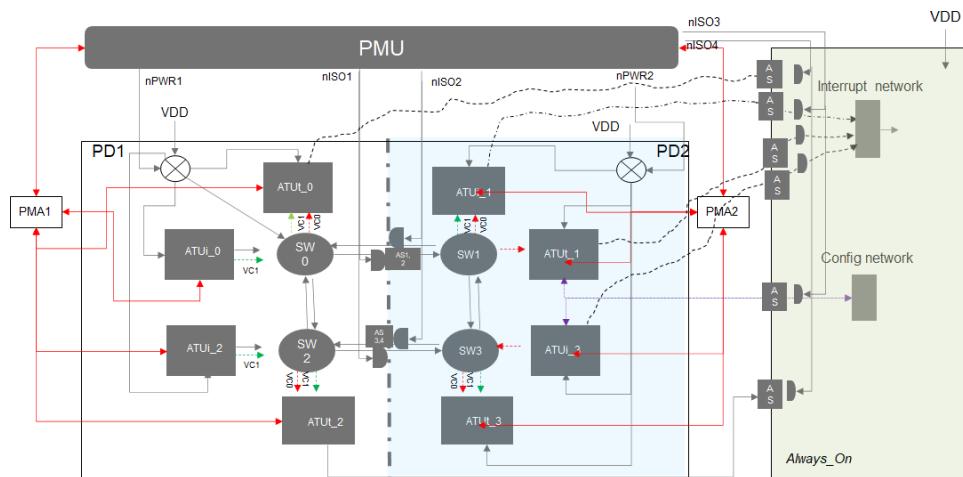
Note: Linux Power Management API, as currently defined, does not have syntax to express “flow” level dependency and only network element level dependency can be described.

## 18.9 Examples:

### 18.9.1 Example 1

Consider the configuration shown in the figure below.

**Figure 18-7 : Example network with power domains, PMU and PMAs**



In the above example, there are two power domains that are controlled by the PMU. Each power domain has an associated PMA. The domains are independently switchable. The power switch in each domain is controlled by a signal from the PMU and so are the isolation cells placed at the output ports of the switches in each domain.

**Table 18-1 Flow-to-Power Domain table**

Flow No.	Src --> Dest	PD1	PD2
0	ATUi_0 --> ATU_t1	Yes	Yes
1	ATUi_0 --> ATU_t2	Yes	No
2	ATUi_0 --> ATU_t3	Yes	Yes
3	ATUi_1 --> ATU_t0	Yes	Yes
4	ATUi_1 --> ATU_t2	Yes	Yes

**Table 18-1 Flow-to-Power Domain table**

5	ATUi_1 -->ATU_t3	No	Yes
6	ATUi_2 -->ATU_t0	Yes	No
7	ATUi_2 -->ATU_t1	Yes	Yes
8	ATUi_2 -->ATU_t3	Yes	Yes
9	ATUi_3 -->ATU_t0	Yes	Yes
10	ATUi_3 -->ATU_t1	No	Yes
11	ATUi_3 -->ATU_t2	Yes	Yes

**Table 18-2 Power state table**

Power States	PD1	PD2
Run	On	On
Off	Off	Off
pRun1	On	Off
pRun2	Off	On

Maestro shall generate a machine readable file with the following format:

FileName:

<Power Domain> :{(flow<sub>0</sub>, Src\_id, Dest\_id), ....., (flow<sub>n</sub>, Src\_id, Dest\_id)} #list of flows in the power domain. The list includes multicast flows

<Power Domain>:{netElementId, ....., netElementId} #list of network elements in the power domain

### **18.9.1.1Powering Off Sequence:**

### **18.9.1.2Power State: Off**

In the Off power state, supply to both domains is turned off. Before the system can enter the Off State the following sequence shall be followed, per this example. Note, the power down sequence is application and system specific and the system designer is in the best position to devise the shut down sequence:

1. The Power Management Application shall quiesce all flows in the system. The flows and their src-dest information can be obtained by reading the file generated by Maestro that contains information about power domains, network elements in the power domain and flows that terminate, originate and crisscross the power domain. That is, Flow 0 to Flow 11 stop sending any new transaction in to the system.
2. The application shall make sure that all outstanding transactions in the system have completed.
3. PMU will inquire from PMA1 and PMA2, if all the elements retain state are idle and can be turned off. If the answer is Yes, the PMU will proceed with asserting the nPWR1 and nPWR2 signal to activate the power switch to turnoff the power supply.

If the answer is that one of the elements is busy, then the PMU will abort and try later.

### **18.9.1.3pRun1:**

In the pRun1 state, only PD1 is On. PD2 is the Off state. Before the system can enter the pRun1 state the following steps shall be taken:

1. Power Management Application shall quiesce all flows that originate, terminate or criss-cross PD2. This information can be read from the file that Maestro spits out. The application shall quiesce all flows in the table above that have source or destination that lie in PD2 and also flows that crisscross PD2.
2. All outstanding transactions for flows that terminate/originate in PD2 or crisscross PD2 should be completed
3. PMU will enquire from PMU2 if the network elements in PD2 can be turned off. If so, the PMU will assert nPWR2 signal and also the nSIO2 signal to turn off power and activate the isolation cell.

### **18.9.1.4pRun2:**

In the pRun2 state, only PD2 is On. PD1 is the Off state. Before the system can enter the pRun2 state the following steps shall be taken:

1. Power Management Application shall quiesce all flows that originate, terminate or criss-cross PD1, as captured in the file spit out by Maestro. The application shall quiesce all flows in the table above that have source or destination that lie in PD1 and also flows that crisscross PD1.
2. All outstanding transactions for flows that terminate/originate in PD2 or crisscross PD1 should be completed
3. PMU will enquire from PMU1 if the network elements in PD1 can be turned off. If so, the PMU will assert nPWR1 signal and also the nSIO1 signal to turn off power and activate the isolation cell.

### **18.9.1.5UPF File:**

The RTL provider creates the RTL and then constraints the RTL with the associated UPF file. That is the UPF file provides the low power constraints on the RTL. The user of this IP then builds the configuration around those UPF constraints and verifies it. The UPF file example in this document will only describe these two aspects and will not delve into implementation details.

The UPF will declare the following:

1. Power Domains which can be element based on configuration based. The IP provider needs to only provide power domains that cannot be further broken down.
2. The state that needs to be retained during shutdown if any.
3. Signals that need to be isolated.
4. Legal power states and sequencing between them without prescribing voltages.

Note: there will three power domains: PD1, PD2 and an *always-on* domain that contains the Configuration and Interrupt network.

For the example configuration shown above the UPF file should contain the following:

1. Create Power domains:

- a. create\_power\_domain PD1 -elements {ATUi\_0 ATUi\_2 ATUt\_0 ATU\_t2 SW0 SW2 AS1 AS2} -shutoff\_condition {!PMU/nPWR1}
- b. create\_power\_domain PD2 -elements {ATUi\_1 ATU\_i3 ATUt\_1 ATUt\_3 SW1 SW3 AS3 AS4} -shutoff\_condition {!PMU/nPWR2}
- c. create\_power\_domain "Always-On" -elements {elements of the config and interrupt network} -supply {VDD}

2. Retention requirements:

- a. None

3. Isolation Requirements:

- a. set\_isolation iSO1 - domain PD1 -allies\_to both - clamp\_value 0 -isolation\_signal PMU/nISO1 -location self
- b. set\_isolation iSO2 -domain PD2 -applies\_to both -clamp\_value 0 -isolation\_signal PMU/nISO2 -location self
- c. set\_isolation iSO3 -domain Always-On --elements { Asynch adapters connected to PD1 -clamp\_value 0 -isolation\_signal PMU/nISO3 -location self}
- d. set\_isolation iSO4 -domain Always-On --elements { Asynch adapters connected to PD2 -clamp\_value 0 -isolation\_signal PMU/nISO4 -location self}

4. Add Power State:

- a. add\_power\_state PD1 -domain
  - ♦ -state {Run -logic\_expr {swVDD == VDD}}
  - ♦ -state {Off -logic\_expr {swVDD == 0}}
- b. add\_power\_state PD2 -domain
  - ♦ -state {Run -logic\_expr {primary == VDD}}
  - ♦ -state {Off -logic\_expr {primary == 0}}
- c. add\_power\_state Always\_On
  - ♦ -state {On -logic\_expr{primary ==VDD}}

5. Power Switch:

- a. create\_power\_switch swPD1 - domain PD1 -output\_supply\_port {swVDD} - input\_supply\_port {pd1VDD} -control\_port {PMU/nPWR1} -on\_state {On pd1VDD {PMU/nPWR1}} -off\_state{ Off {!PMU/nPWR1}}
- b. create\_power\_switch swPD2 - domain PD2 -output\_supply\_port {swVDD} - input\_supply\_port {pd2VDD} -control\_port {PMU/nPWR2} -on\_state {On pd2VDD {PMU/nPWR2}} -off\_state{ Off {!PMU/nPWR2}}

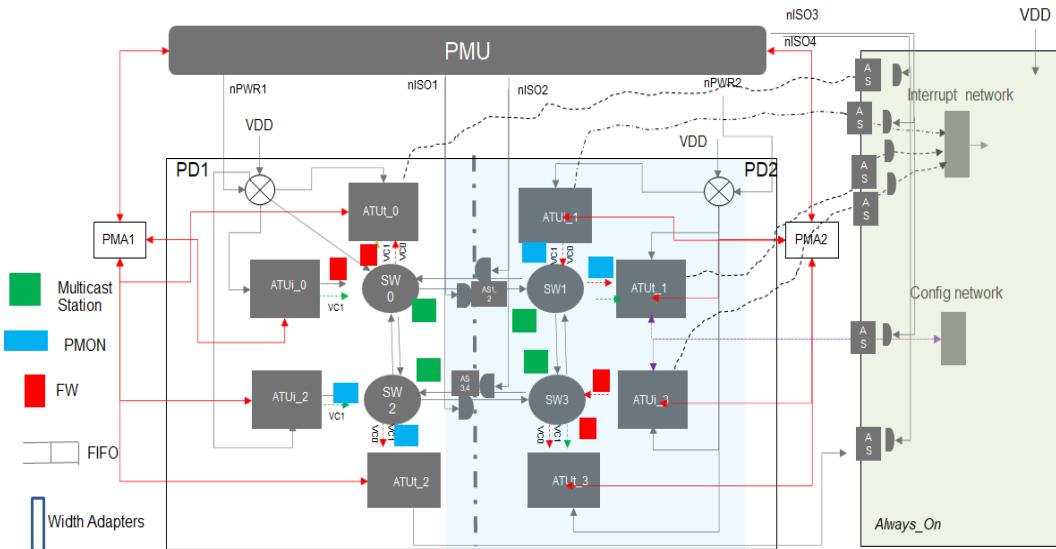
6. Level Shifters

- a. set\_level\_shifter -domain PD1 -elements SW1/output\_port SW3/output\_port -location self
- b. set\_level\_shifter -domain PD2 -elements SW0/output\_port SW2/output\_port -location self
- c. set\_level\_shifter -domain Always\_On applies\_to both -location self

## 18.9.2 Example 2:

In the example illustrated in below, more network elements have been added. It should be noted, that other than the flow table not much will change in the UPF file structure from the previous example. The elements added do not add anything to the complexity of design other than that to the flow table. Adding power domains and power states would add to the complexity of the design from a power management perspective. The network consists of three power domains: PD1, PD2 and the Always-On domain. PD1 and PD2, each contain the following network elements: ATUi, ATUt, PMON prob, FW, FIFO, Width adapter and Multicast Stations. The network uses 3 VCs to carry traffic. One VC is dedicated for multicast traffic and the other two are used to carry priority traffic.

**Figure 18-8 Complex Network with 3 power domains: PD1, PD2 and Always-On.**



**Figure 18-9 Flow to Power Domain Mapping**

Flow No.	Src --> Dest	PD1	PD2	VCs
0	ATUi_0 --> ATU_t1	Yes	Yes	0
1	ATUi_0 --> ATU_t2	Yes	No	0
2	ATUi_0 --> ATU_t3	Yes	Yes	0
3	ATUi_1 --> ATU_t0	Yes	Yes	1
4	ATUi_1 --> ATU_t2	Yes	Yes	1
5	ATUi_1 --> ATU_t3	No	Yes	1
6	ATUi_2 --> ATU_t0	Yes	No	0
7	ATUi_2 --> ATU_t1	Yes	Yes	0

8	ATUi_2 --> ATU_t3	Yes	Yes	0
9	ATUi_3 --> ATU_t0	Yes	Yes	1
10	ATUi_3 --> ATU_t1	No	Yes	1
11	ATUi_3 --> ATU_t2	Yes	Yes	1

**Table 18-3 Multi-flow table.**

Multi-Flow	Src->Dest	PD1	PD2	VC
12	ATUi_0 --> {ATUt_1, ATUt_3}	Yes	Yes	2
13	ATUi_1 --> {ATUt_0, ATUt_2}	Yes	Yes	2
14	ATUi_2 --> {ATUt_0, ATUt_1, ATUt_3}	Yes	Yes	2

From the tables above it is clear that in order to turn off any of the power domains, the flows originating or terminating or criss-crossing the domain would have to be turned off. It should also be clear from the above that any of the domains cannot be turned off until all the multicast flows are quiescent.

### 18.9.2.1 UPF File:

The UPF file for this example is similar to the previous example, except for the elements contained in each domain.

The RTL provider creates the RTL and then constraints the RTL with the associated UPF file. That is the UPF file provides the low power constraints on the RTL. The user of this IP then builds the configuration around those UPF constraints and verifies it. The UPF file example in this document will only describe these two aspects and will not delve into implementation details. As in the previous example, there are three domains in the system.

For the example configuration shown above the UPF file should contain the following:

1. Create Power domains:

- a. create\_power\_domain PD1 -elements {ATUi\_0 ATUi\_2 ATUt\_0 ATU\_t2 SW0 SW2 MC0 MC2 PMON0 PMON2 FIFO0 FIFO2 WA0 WA2 AS1 AS2} -shutoff\_condition {!PMU/nPWR1}
- b. create\_power\_domain PD2 -elements {ATUi\_1 ATU\_i3 ATUt\_1 ATUt\_3 SW1 SW3 MC1 MC3 PMON1 PMON3 FIFO1 FIFO3 WA1 WA2 AS3 AS4} -shutoff\_condition {!PMU/nPWR2}
- c. create\_power\_domain “Always-On” -elements {elements of the config and interrupt network} -supply {VDD}

2. Retention requirements:

- a. None

3. Isolation Requirements:

- a. set\_isolation iSO1 - domain PD1 -allies\_to both - clamp\_value 0 -isolation\_signal PMU/nISO1 - location self
- b. set\_isolation iSO2 -domain PD2 -applies\_to both -clamp\_value 0 -isolation\_signal PMU/nISO2 -location self

- c. set\_isolation iSO3 -domain Always-On --elements { Asynch adapters connected to PD1 -clamp\_value 0 -isolation\_signal PMU/nISO3 -location self}
  - d. set\_isolation iSO4 -domain Always-On --elements { Asynch adapters connected to PD2 -clamp\_value 0 -isolation\_signal PMU/nISO4 -location self}
4. Add Power State:
- a. add\_power\_state PD1 -domain
    - ♦ -state {Run -logic\_expr {swVDD == VDD}}
    - ♦ -state {Off -logic\_expr {swVDD == 0}}
  - b. add\_power\_state PD2 -domain
    - ♦ -state {Run -logic\_expr {primary == VDD}}
    - ♦ -state {Off -logic\_expr {primary == 0}}
  - c. add\_power\_state Always\_On
    - ♦ -state {On -logic\_expr{primary ==VDD}}
5. Power Switch:
- a. create\_power\_switch swPD1 - domain PD1 -output\_supply\_port {swVDD} - input\_supply\_port {pd1VDD} -control\_port {PMU/nPWR1} -on\_state {On pd1VDD {PMU/nPWR1}} -off\_state{ Off {!PMU/nPWR1}}
  - b. create\_power\_switch swPD2 - domain PD2 -output\_supply\_port {swVDD} - input\_supply\_port {pd2VDD} -control\_port {PMU/nPWR2} -on\_state {On pd2VDD {PMU/nPWR2}} -off\_state{ Off {!PMU/nPWR2}}
6. Level Shifters
- a. set\_level\_shifter -domain PD1 -elements SW1/output\_port SW3/output\_port - location self
  - b. set\_level\_shifter -domain PD2 -elements SW0/output\_port SW2/output\_port - location self
  - c. set\_level\_shifter -domain Always\_On applies\_to both -location self

Note as mentioned above other than including the elements in the power domain nothing else changes from example 1.

The configuration in Example 2 is detailed in the spreadsheet (sheetname: “configuration 6\_5\_3”) at the following location:

[https://confluence.arteris.com/pages/viewpage.action?spaceKey=ENGR&title=VC\\_Configs+for+the+Verification+Table](https://confluence.arteris.com/pages/viewpage.action?spaceKey=ENGR&title=VC_Configs+for+the+Verification+Table)

### 18.9.3 Error Management:

Error management is supported via timeout at the ATUi. If a transaction is sent to the domain that is power off, the transaction will not be responded to. The timer will time out and an interrupt will be raised.

Since error management is handled through raising an interrupt, the DV bench would have to use the interrupt network to identify interrupt raised and use the configuration network to read the interrupt error register. In case the timeout happens, the timeout error register has to be read. The timeout error register also identifies the target ID of the target which caused the timeout.

Note on Response Network is the mirror image of the request network and mirrored response network elements reside in the same power domain as the request network elements.

## 18.10 Interrupt and Configuration Network

The interrupt and configuration network is needed for the reason described above. However, the interrupt and configuration network should be in the “*always-on*” domain.

## 18.11 Clock Gating

Symphony units will have three levels of clock gating. The third level of clock gating is done at the flop level. Any combination of three or more flops should be gated. This gating can be done manually or can be inferred by the synthesis tool. Note that proper enable signals should be provided to all flops which are to be gated. The second level of clock gating is done at the unit level and the first level of clock gating is done at the clock sub domain level. Clock sub domain and unit level clock gating conditions are described in the power management state section.

### 18.11.1 Internal Clock gating rules

At the architecture level, gating rules will remain unspecified and left as a detail for microarchitecture. It can be envisioned that a small portion of logic can remain clocking inside a clock domain when the domain is in the off (sleep ready) state so that the Clock and Power Management Agents can function.

## 18.12 Clock Interface

A clock interface is the set of control signals, reset signals and clock signals that control the clocking and reset of a clock sub domain. The exact signaling of a clock interface is controlled by microarchitecture. A single clock interface shall be associated with a clock sub domain.

---

Implementation

---

It is envisioned that there is a standard Clock Interface definition that is used throughout Arteris for all projects the use the same software tooling as Symphony Hardware.

---

## 18.13 Power and Clock Management Domain Interface

### 18.13.1 Interface Rules

There will be one interface per clock domain. Power domains will not have an interface directly associated with it. Control over a power domain is achieved by controlling all the clock domains enclosed by a power domain.

There can be only one interface or many interfaces to a Symphony NOC.

Multiple Interfaces of a single type (power or clock) can be aggregated to a single interface to support hierarchical creation of blocks.

#### 18.13.1.1 Switchable vs. Non-switchable domains

If a domain does not need to be put to sleep (non-switchable), than the domain does not need an interface associated with it.

If a power domain is can be put to sleep (switchable) then all the clock domains enclosed shall be switchable.

### 18.13.2 Interface definition for Clock Domains (Slave view)

Clock domain control can be done through a Q-Channel or P-Channel interface. Below are definitions of the signals unique to each interface and the common signals that will be present in either interface.

#### 18.13.2.1 Q-Channel Interface

##### 18.13.2.1.1 Input CREQn

Asserting Is not allowed if either CACCEPTn or CDENY is asserted.

Deasserting is not allowed unless either CACCECTn or QDENY is asserted.

When domain is in active state and both CACCEPTn and CDENY are both deasserted, asserting indicates PMU is requesting domain to enter sleep ready state.

When domain is in sleep ready state and CACCEPTn is asserted, deasserting CREQn indicates PMU is requesting domain to exit sleep ready state.

When domain is in active state, CREQn is asserted, and CDENY is asserted, deasserting indicates PMU is removing request for domain to enter sleep ready state.

##### 18.13.2.1.2 Output CACCEPTn

Should never be asserted at the same time as CDENY.

Asserting is not allowed unless CREQn is asserted and CDENY is deasserted.

Deasserting is not allowed unless CREQn is deasserted.

Asserting while CREQn is asserted indicates the domain has entered the sleep ready state. Clocks can be stopped or started.

Deasserting while CREQn is deasserted indicates the domain has entered the active state.

### **18.13.2.1.3 Output CDENY (optional)**

Should never be asserted at the same time as CACCEPTn.

Asserting is not allowed unless CREQn is asserted and CACCEPTn is deasserted.

Asserting while CREQn is asserted indicates that the domain is denying the request by the PMU for the domain to enter the sleep ready state.

Deasserting while the CREQn is deasserted indicates that the domain is ready to receive another request to enter the sleep ready state.

## **18.13.2.2 P-Channel Interface**

### **18.13.2.2.1 Input CREQ**

Asserting is not allowed if either CACCEPT or CDENY are asserted.

Deasserting is not allowed unless CACCEPT or CDENY is asserted.

CREQ being asserted indicates a request for the slave to move into the state indicated on CSTATE.

When CACCEPT asserts the slave has successfully moved to CSTATE.

When CDENY asserts the slave indicates that it cannot move to CSTATE and will revert to the previous state.

### **18.13.2.2.2 Input CSTATE**

CSTATE indicates the power state being requested. 0 is Sleep Ready and 1 is Active.

CSTATE can only change when

- CACCEPT, CDENY, and CREQ are asserted
- CDENY and CREQ are asserted, CACCEPT is deasserted

When a request is denied CSTATE MUST revert back to the previous state before CREQ is deasserted.

### **18.13.2.2.3 Input CACCEPT**

CACCEPT cannot be asserted the same time as CDENY

CACCEPT can only assert when CREQ is asserted and CDENY is deasserted.

CACCEPT can only deassert when CREQ and CDENY are deasserted.

CACCEPT asserting indicates that the domain has successfully entered the state indicated by CSTATE.  
 Deasserting means the domain is ready to receive another request.

#### **18.13.2.2.4 Input CDENY (optional)**

Should never be asserted the same time as CACCEPT.

CDENY can only assert when CREQ is asserted and CACCEPT is deasserted.

CDENY can only deassert when CREQ and CACCEPT are deasserted.

Asserting means that the domain denied the request and will move back to the previous state.

Deasserting means the domain is ready to receive another request.

#### **18.13.2.3 Common Interface**

##### **18.13.2.3.1 Input CACTIVE (optional)**

Domain status of internal state. Is an indication to the PMU that the domain is actively working when asserted.

CACTIVE should never assert when the domain is in sleep ready state.

##### **18.13.2.3.2 Input CWAKEUP[N-1:0] (optional)**

A vector of signals coming from blocks outside of the domain that are requesting that blocks inside the domain to exit the sleep ready state.

#### **18.13.3 Interface definition for Power Domains (Slave view)**

Power domain control can be done through a Q-Channel or P-Channel interface. Below are definitions of the signals unique to each interface and the common signals that will be present in either interface.

##### **18.13.3.1 Q-Channel Interface**

###### **18.13.3.1.1 Input PREQn**

Asserting is not allowed if either PACCEPTn or PDENY is asserted.

Deasserting is not allowed unless either PACCECPn or PDENY is asserted.

When domain is in active state and both PACCEPTn and PDENY are both deasserted, asserting indicates PMU is requesting domain to enter sleep ready state.

When domain is in sleep ready state and PACCEPTn is asserted, deasserting PREQn indicates PMU is requesting domain to exit sleep ready state.

When domain is in active state and PDENY is asserted, deasserting indicates PMU is removing request for domain to enter sleep ready state.

### **18.13.3.1.2 Output PACCEPTn**

Should never be asserted at the same time as PDENY.

Asserting is not allowed unless PREQn is asserted and PDENY is deasserted.

Deasserting is not allowed unless PREQn is deasserted.

Asserting while PREQn is asserted indicates the domain has entered the sleep ready state. Clocks can be stopped or started.

Deasserting while PREQn is deasserted indicates the domain has entered the active state.

### **18.13.3.1.3 Output PDENY (optional)**

Should never be asserted at the same time as PACCEPTn.

Asserting is not allowed unless PREQn is asserted and PACCEPTn is deasserted.

Asserting while PREQn is asserted indicates that the domain is denying the request by the PMU for the domain to enter the sleep ready state.

Deasserting while the PREQn is deasserted indicates that the domain is ready to receive another request to enter the sleep ready state.

## **18.13.3.2 P-Channel Interface**

### **18.13.3.2.1 Input PREQ**

Asserting is not allowed if either PACCEPT or PDENY are asserted.

Deasserting is not allowed unless PACCEPT or PDENY is asserted.

PREQ being asserted indicates a request for the slave to move into the state indicated on PSTATE.

When PACCEPT asserts the slave has successfully moved to PSTATE.

When PDENY asserts the slave indicates that it cannot move to PSTATE and will revert to the previous state.

### **18.13.3.2.2 Input PSTATE**

PSTATE indicates the power state being requested. 0 is Sleep Ready and 1 is Active.

PSTATE can only change when

- PACCEPT, PDENY, and PREQ are asserted
- PDENY and PREQ are asserted, PACCEPT is deasserted

When a request is denied PSTATE MUST revert back to the previous state before PREQ is deasserted.

### **18.13.3.2.3 Input PACCEPT**

PACCEPT cannot be asserted the same time as PDENY

PACCEPT can only assert when PREQ is asserted and PDENY is deasserted.

PACCEPT can only deassert when PREQ and PDENY are deasserted.

PACCEPT asserting indicates that the domain has successfully entered the state indicated by PSTATE.

Deasserting means the domain is ready to receive another request.

### **18.13.3.2.4 Input PDENY (optional)**

Should never be asserted the same time as PACCEPT.

PDENY can only assert when CREQ is asserted and PACCEPT is deasserted.

PDENY can only deassert when PREQ and PACCEPT are deasserted.

Asserting means that the domain denied the request and will move back to the previous state.

Deasserting means the domain is ready to receive another request.

## **18.13.3 Common Interface**

### **18.13.3.1 Input PACTIVE (optional)**

Domain status of internal state. Is an indication to the PMU that the domain is actively working when asserted.

PACTIVE should never assert when the domain is in sleep ready state.

### **18.13.3.2 Input PWAKEUP[N-1:0] (optional)**

A vector of signals coming from blocks outside of the domain that are requesting that blocks inside the domain to exit the sleep ready state.

## **18.13.4 Example Domain**

Two example systems showing the Clock Domain Interface and different power management blocks are shown below, these blocks are:

Power Management Unit (PMU)

Power Management Agent Master (PMA\_M)

Power Management Agent Slave (PMA\_S)

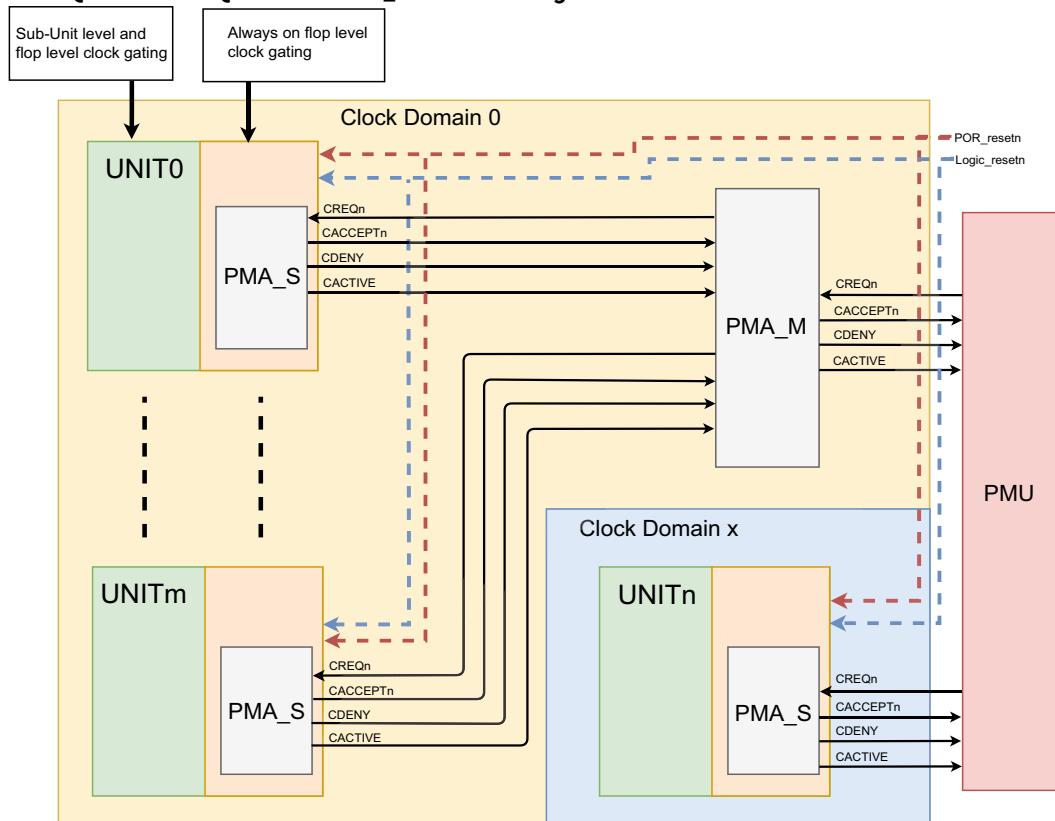
Power Management Unit is not a symphony block; it is a customer implemented block. The purpose of this block is to control and maintain different power domains within symphony. PMU is responsible for both powering down and powering up a power domain. The PMU communicates with symphony using a Q-Channel or P-Channel compatible signal interface.

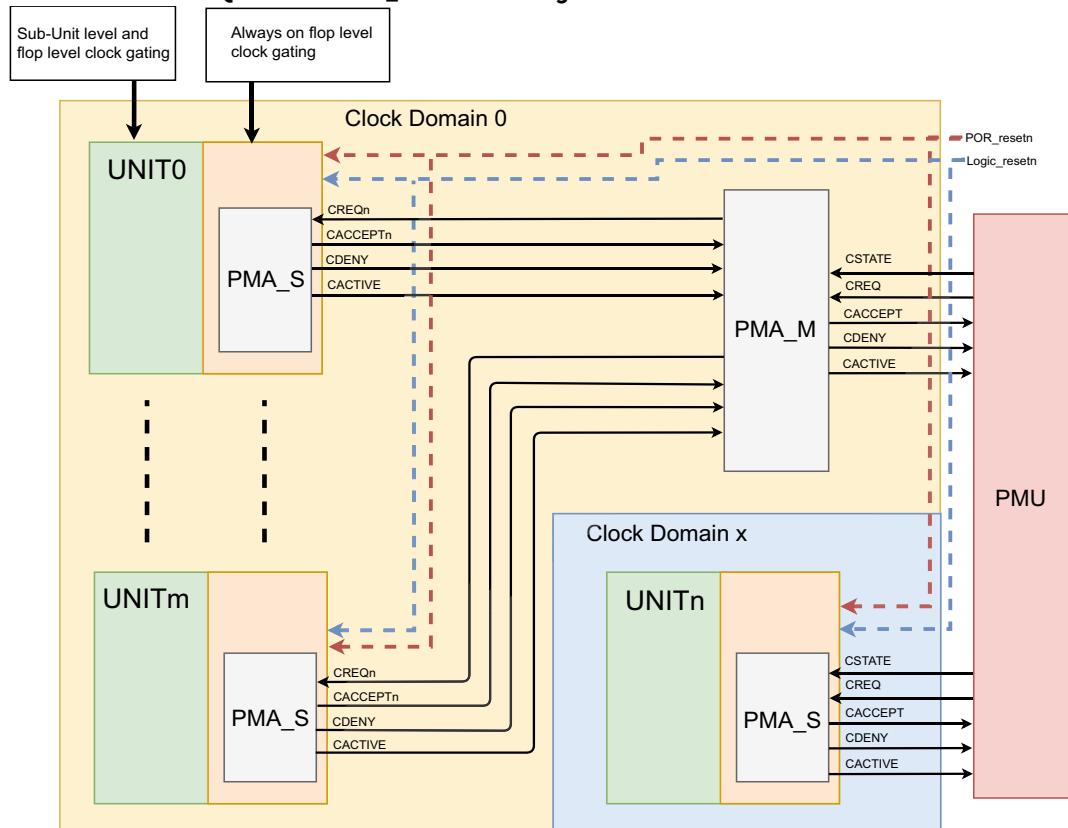
Power Management Agent Slave resides within the symphony unit. It communicates either with PMU or PMA\_M using the a Q-Channel or P-Channel compatible interface. It monitors its host unit and implements the power management state machine. Note that PMA\_S resides in the always on part of the unit as shown in Figure 18-10 on page 277.

Power Management Agent Master communicates with the PMU on the master side and with two or more PMA\_S on the slave side. The PMA\_M can be configured to have a P-Channel on the PMU side and a Q- or P-Channel on the PMA\_S side, or a Q-Channel on the PMU side and a Q-Channel on the PMA\_S side. The purpose of this block is to consolidate the number of power and clock interfaces to PMU. The PMA\_M will accept a power down request only when all the PMA\_S connected to it are in the Sleep Ready state, and it will refuse the request if any of the slaves deny the powerdown request.

Note that PMA\_M may communicate with PMA\_S and PMU which are in different clock domains. To simplify the implantation, it can be presumed that all Q- and P-Channel interfaces for both PMA\_S and PMA\_M are asynchronous and should be synchronized.

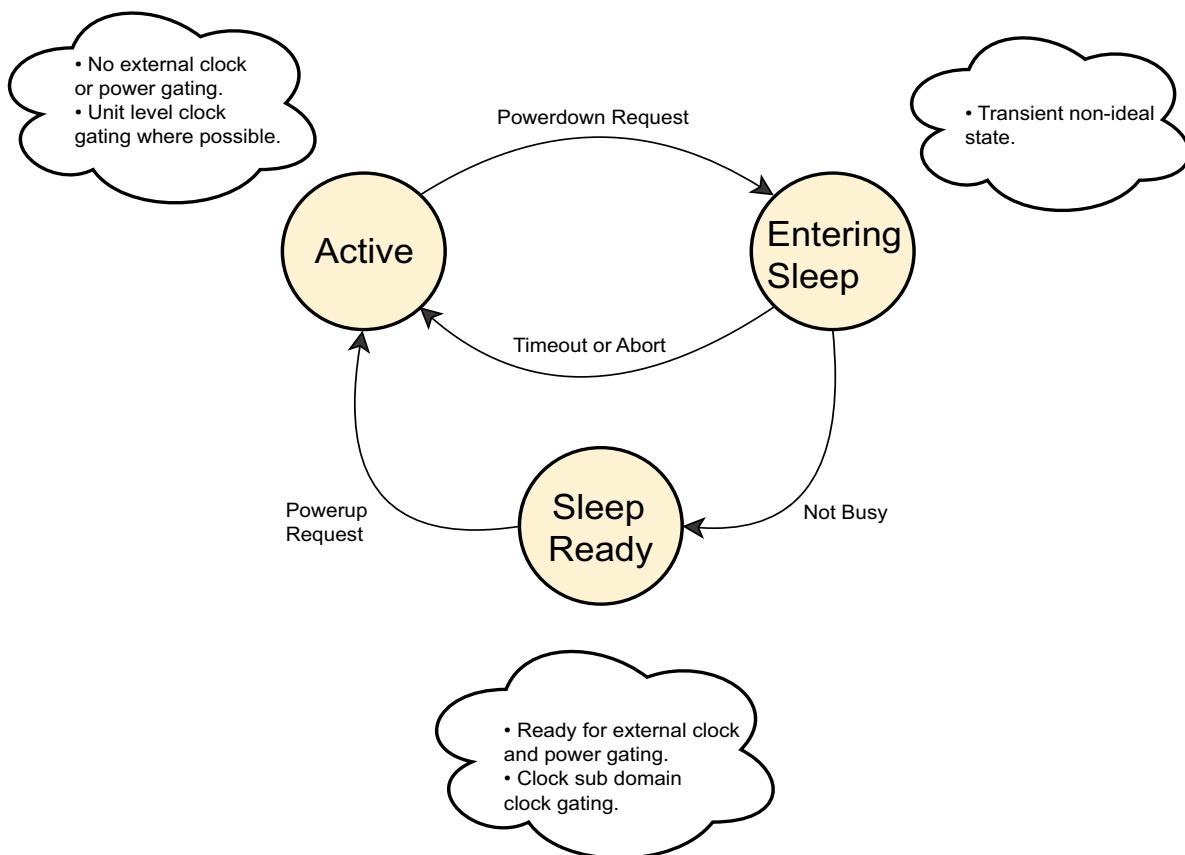
**Figure 18-10 Q-Channel to Q-Channel PMU\_M Power Management Interface**



**Figure 18-11 P-Channel to Q-Channel PMU\_M Power Management Interface**

## 18.14 Power Management States

Symphony units should support three power management states Active, Entering Sleep, and Sleep Ready for both power and clock domains. These states apply to both the Q- and P-Channel Interfaces. These states are shown in Figure 18-12 on page 279.

**Figure 18-12 Symphony Power/Clock Domain States**

**Active:** In this state either a part or all of the unit is processing information. Units can be clock gated in this state. If power/clock are turned off in this state, the unit will return in an undetermined state.

**Entering Sleep:** In this state the unit got a power down request and is attempting to go to the Sleep Ready state. The unit will go to the Sleep Ready state once busy is deasserted or the unit will deny the powerdown request and go to the Active state if abort is asserted. If power/clock are turned off in this state, the unit will return in an undetermined state

**Sleep Ready:** In this state the unit is idle and not waiting for any event. The clock sub domain itself should be clock gated. The power/clock can be turned off in this state. The unit may retain its configuration when the clock is turned off but the power is maintained at a level that maintains state.

When a power down request is first sent to the unit the logic inside other than the PMA will remain active, but an internal sleep signal will be asserted that prevents slave interfaces from accepting new requests. An exception to this is slave interfaces that can auto-wake the domain. During this time the PMA block watches the interrupts and busy signals of the blocks to determine when it can complete the request back to the PMU. If configured to do so, if the blocks don't clear all interrupts and have all blocks go not busy in a fixed amount of time, the PMA will deny the request to the PMU. In addition the PMA can be configured to automatically deny the request to the PMU if an interrupt is not cleared and/or a block is busy during this time. Whether a signal holds off completing the request or immediately denies a request is determined by its connection to either the busy or abort input to the PMA.

## 18.14.1 Off State Behavior

Off State applies to both Power and Clock Domains and indicates that the domain is not being clocked or does not have power so the block is considered in the Sleep Ready State. You can be in the Sleep Ready State and still have clocks toggling or power applied.

When an initiator is tied directly to the domain that is off, A signal can indicate that the domain is off.

## 18.14.2 Retention

Retention is when you turn off clocks or you go through the sequence of turning off power, but don't actually turn the power off. When a block is in the off state, but you don't actually remove power, you can resume functional operation by deasserting your power-down request and then the domain will deassert its accept and resume in functional mode. If you want to return the domain to its reset state but not reconfigure the domain, there will be two resets to aid this. One reset will tie to configuration and is used to reset the configuration to the default state and the other is used to reset everything else.

Some customers will lower the power to save power but it's up to them to characterize how low they can lower the power before they lose internal state of the domain.

## 18.14.3 Auto Wakeup

When a request interface into the domain has been marked as auto-wakeup, it no longer provides an "off" signal and its valid signal is used to generate an activate signal that is driven to the PMU. So, for a single domain, there is a vector of PWAKEUP or QWAKEUP signals; one for each interface that can cause an auto-wakeup.

The ultimate source of these signals is not architected. This is because for domains that power down, the logic can't be in the domain when the power is turned off. In a clock domain, it's fairly easy to only clock a small domain of logic in the clock off state, but for consistency reasons it may be decided to handle both domains the same.

The protocol is that the domain will not really auto wakeup. The power down request must be removed by the PMU and only then can the logic resume normal function after it has deasserted its accept signal.

## 18.14.4 Abort

The proper method to abort by both the PMU and the PMA is described in the Sequences section.

## 18.14.5 Auto Configuration on Wakeup from Power Off

An automatic configuration by hardware of a block that has had the power turned off is not supported.

## 18.15 Sequences

Failure to follow proper sequencing will result in undefined behavior which includes the total functional failure of the Symphony NOC. Guaranteed recover can only be achieved through a POR sequence. These sequences work for interfaces into a PMA\_S or PMA\_M block.

## 18.15.1 Active to Sleep Ready Sequence Clock Domains

### 18.15.1.1 Q-Channel

1. CREQn asserts (domain in Entering Sleep State)
2. CACCEPTn asserts (domain in Sleep Ready State)
3. Turn clock off

### 18.15.1.2 P-Channel

1. CSTATE is set to 0
2. CREQ asserts (domain in Entering Sleep State)
3. CACCEPT asserts
4. CREQ deasserts
5. CACCEPT deasserts (domain in Sleep Ready State)
6. Turn clock off

## 18.15.2 Active to Sleep Ready Sequence Power Domains

### 18.15.2.1 Q-Channel

1. Move clock domains associated with power domain to Sleep Ready.
2. Turn power off or lower power rails

### 18.15.2.2 P-Channel

1. Move clock domains associated with power domain to Sleep Ready.
2. Turn power off or lower power rails

## 18.15.3 Active to Sleep Ready Abort Sequence, internally aborted Clock Domains

### 18.15.3.1 Q-Channel

1. CREQn/PREQn asserts (domain in Entering Sleep State)

- 
- 2. CDENY/PDENY asserts
  - 3. CREQn/PREQn deasserts
  - 4. CDENY/PDENY deasserts (domain in Active State)

### **18.15.3.2 P-Channel**

- 1. CSTATE /PSTATE is set to 0
- 2. CREQ /PREQ asserts (domain in Entering Sleep State)
- 3. CDENY/PDENY assert
- 4. CREQ /PREQ deasserts
- 5. CDENY/PDENY deasserts (domain in Active State)

## **18.15.4 Active to Sleep Ready Abort Sequence, externally aborted Clock Domains**

An external abort can only be sent on an interface directly to a PMA\_S.

### **18.15.4.1 Q-Channel**

- 1. PREQn or QREQn asserts
- 2. PREQn or QREQn holds for at least 5 slowest clocks at boundary of block
- 3. PREQn or QREQn deasserts

### **18.15.4.2 P-Channel**

- 1. PSTATE or QSTATE is set to 0
- 2. PREQ or QREQ asserts (domain in Entering Sleep State)
- 3. PSTATE or QSTATE is set to 1
- 4. Hold for at least 5 slowest clocks at boundary of block
- 5. PREQ or QREQ deasserts
- 6. Hold for at least 5 slowest clocks at boundary of block (domain in Active State)

## **18.15.5 Sleep Ready to Active Sequence Clock Domains**

### **18.15.5.1 Q-Channel**

1. Turn clocks on
2. CREQn deasserts
3. CACCEPTn deasserts (domain in Active State)

### **18.15.5.2 P-Channel**

1. Turn clocks on
2. CSTATE is set to 1
3. CREQ asserts
4. CACCEPT asserts
5. CREQ deasserts
6. CACCEPT deasserts (domain in Active State)

## **18.15.6 Sleep Ready to Active Sequence Power Domains state not retained**

### **18.15.6.1 Q-Channel**

1. Assert all resets
2. Turn power on
3. Wait for power to stabilize
4. Turn clocks on
5. Deassert all resets
6. PREQn deasserts for all clock domains enclosed in power domain
7. PACCEPTn deasserts for all clock domains enclosed in power domain (domain in Active State)

### **18.15.6.2 P-Channel**

1. Assert all resets
2. Turn power on
3. Wait for power to stabilize
4. Turn clocks on.
5. Deassert all resets
6. PSTATE is set to 1 for all clock domains enclosed in power domain
7. PREQ asserts for all clock domains enclosed in power domain

- 
8. PACCEPT asserts for all clock domains enclosed in power domain
  9. PREQ deasserts for all clock domains enclosed in power domain
  10. PACCEPT deasserts for all clock domains enclosed in power domain (domain in Active State)

## **18.15.7 Sleep Ready to Active Sequence Power Domains state retained**

### **18.15.7.1 Q-Channel**

1. Assert non-configuration resets (optional)
2. Raise Power rails
3. Wait for power to stabilize
4. Move clock domains associated with power domain from Sleep Ready to Active

### **18.15.7.2 P-Channel**

1. Assert non-configuration resets (optional)
2. Raise Power rails
3. Wait for power to stabilize
4. Move clock domains associated with power domain from Sleep Ready to Active

## **18.15.8 PMA Reset to Active State Sequence**

### **18.15.8.1 Q-Channel**

1. Reset is asserted to the PMA
2. Deassert CREQn/PREQn for at least 5 slowest clocks at boundary of block
3. Deassert reset to PMA

### **18.15.8.2 P-Channel**

1. Reset is asserted to the PMA

2. Assert CSTATE/PSTATE to 1 or at least 5 slowest clocks at boundary of block
3. Deassert reset to PMA

## 18.15.9 PMA Reset to Sleep Ready State Sequence

### 18.15.9.1 Q-Channel

1. Reset is asserted to the PMA
2. Assert CREQn/PREQn for at least 5 slowest clocks at boundary of block
3. Deassert reset to the PMA

### 18.15.9.2 P-Channel

1. Reset is asserted to the PMA
2. Deassert CSTATE/PSTATE to 0 or at least 5 slowest clocks at boundary of block
3. Deassert reset to PMA

## 18.15.10 Auto Wakeup Sequence Power and Clock Domains

1. One bit in CWAKEUP or PWAKEUP bus asserts
2. PMU determines it should wake up the domain
3. Follow steps to transition domain from Sleep Ready to Active

## 18.15.11 Sequence Duration

Sequence Duration can be determined by summing the duration that it takes to do each of the steps in the sequence. These durations are implementation dependent and outside the scope of this specification.

## 18.16 Dynamic Voltage and frequency Scaling

This a physical design and standard cell library dependent feature. Symphony is architected to make sure that it does not impose any additional restrictions on implementing DVFS as long as the proper power and clock domains are defined. Note that all DVFS rules and restrictions as per the standard cell library used apply.

## 18.17 Future Work

Future work will include the following:

- ▶ Power aware routing: That is routing of flows in a manner that reduces the overall power consumption of the NoC.
- ▶ Power Domain-Aware routing: Routing of flows in a manner that minimizes power domain crossings and allows for maximum number of power domains to be turned off

# Performance Monitoring (PMON)

This section defines the performance monitoring architecture in Symphony. Note that the performance monitoring and debug/trace functionality (described in Section Debug and Trace on page 293) have common concepts and that elements of one can feed into the functionality of the other.

## 19.1 Definition of Performance Monitoring

The following attributes define performance monitoring in the context of Symphony:

1. Capability to measure and access identified events in the Symphony interconnect
2. Means to count the number of events as well as to be able to track the period of time that events are counted over
3. Means to be able to read accumulated performance measurements through Symphony's configuration interface
4. Capability to generate interrupts based on measurement results
5. Capability to put performance-related events on the trace bus and/or connect to the trigger generation module

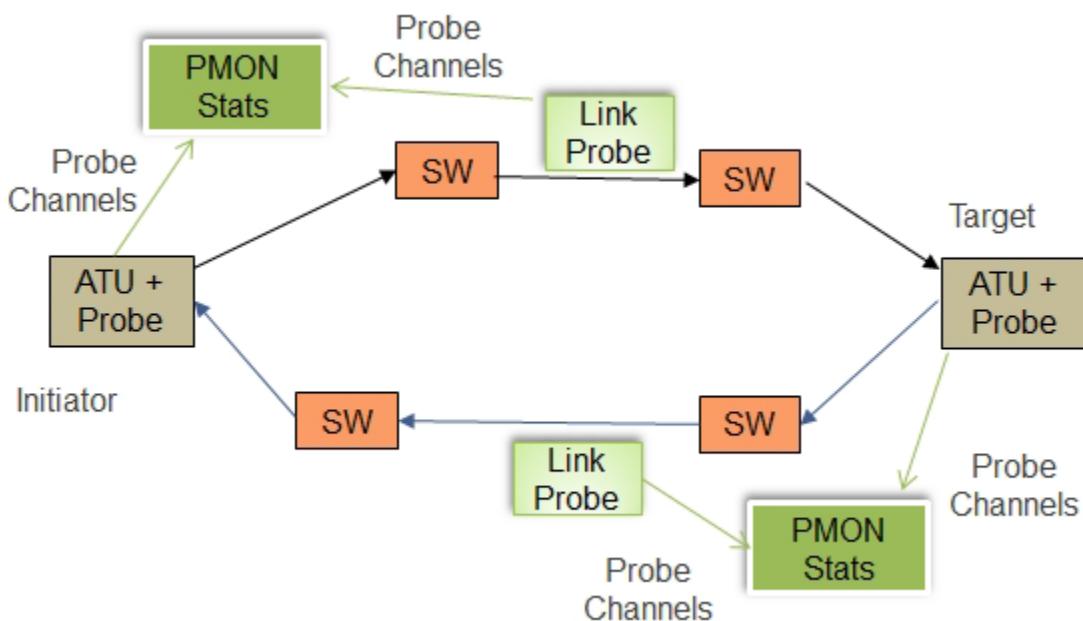
## 19.2 PMON Components Enable PMON Functionality

PMON blocks are optional and are provisioned by the user at design time. There are two major PMON components:

1. PMON probes – are added to a Symphony component when a component has an event (or events) that the user desires to monitor from a performance perspective. A probe may carry an event and, if configured for it, information associated with an event. An event is a pulse created as a result of a defined action having taken place. In general, an event is transported on a single wire probe. However, there are some cases where an event is a count of something that will be accumulated – the “pulse” in this case is a non-zero value encoded on a bus. The probe function includes configuration fields that are used to select and filter events in the component by qualifying the event against a specified criterion. The events are then exported out of the component on the probe.
2. Each Symphony component has a list of events that are relevant from a performance monitoring point of view. These are defined as “raw” events. A raw event is one that

- is not qualified based on some attribute field. A write request out of an ATU is a raw event.
3. PMON probes are placed on a component when that component has events that a user wants to monitor to measure performance
  4. A selection field in a PMON probe configuration register is used to select which raw events in the component to filter and export to a standard PMON statistical block (described below)
  5. Filter control fields in a PMON probe configuration register define the type of filtering to perform on the raw events and information (attribute) needed for the filtering operation. For example, a raw ATU write select may be filtered based on which destination the write is going to. The filter field instructs the PMON logic to filter based on destination and provides the destination value to filter for.
  6. The filtered event is exported out of the component on probe “channels.” There are typically many events that we may want to count (a large menu). Sending all of the events to the resource that does the counting (the PMON statistics block) is expensive. Thus, muxing is done in the source component to reduce the number of events sent to the stats block over the probe channels to match the counting resources provisioned in the stats block. The number of counters in the stats block reflects the number of statistics they can make in parallel (at the same time).
  7. PMON statistical block – this block contains logic to accumulate filtered PMON events from one or more PMON probes. The PMON statistical block will have the following elements:
    - a. Timer with a scalable clock tick
    - b. A configurable number of counters. One counter counts one filtered PMON event. Note that raw events can be counted by not specifying any filtering to be performed on it.
    - c. Control registers needed to select PMON events since multiple filtered PMON events can feed into one PMON statistical block
    - d. Configuration access to control registers, timer register, and counter registers.
    - e. Counter/timer enable mechanism can be either generated directly from a local control register or from a global trigger
    - f. The stats block will be able to generate interrupts. Interrupt generation can be done based on the following (this is not a complete list):
      - ♦ Counter/timer saturation
      - ♦ Counter/timer equal to or greater a threshold value
    - g. Stat blocks themselves can have probes added to them. This allows us to cascade stat blocks, or feed events generated by a timer/counter in a stats block to a different counter in the same stats block.

PMON components are shown in a Symphony interconnect in Figure 19-1 on page 289.

**Figure 19-1 PMON Components in a Symphony Interconnect**

## 19.3 PMON is Optional

The provisioning and use of PMON components is completely optional. The customer is not required to have any PMON-related circuitry in their interconnect. We may decide to put a minimum amount of PMON support in every interconnect. Since the PMON capability described here is enabled by parameter (configurable at design time based on customer input and/or what can be called an Arteris “we want this function in there” control file), we can have a minimum content policy that changes over time. How much PMON to put in and whether to put in any at all can change as the product matures.

A user can either choose to have dedicated statistics per VN (so that the statistics on one VN are not visible to other VNs) or to filter events based on VN, so that a statistics module can be shared across VNs.

The general use case is that a customer will determine what blocks they want to perform PMON on. They will specify how many statistics they need to count in parallel, what kind of filtering they want to do, and how they want to initiate PMON statistics gathering. Based on that input and a similar Arteris PMON control file, our SW will set probe parameters and instantiate statistic blocks when building the interconnect.

## 19.4 Example Events

The following is an example list of events that are visible to PMON (not all components that support PMON events are listed below):

1. ATU events: The ATU is a special component from a PMON point of view because of the amount of filtering that can occur. Filtered events from ATUs can either feed PMON statistical blocks instantiated directly in that ATU (enabled by parameter) or can be exported to a statistical block that resides elsewhere.

- a. Write requests received from native interface
- b. Write requests issued to network
- c. Error responses received
- d. Number of clocks when a write request is being stalled and there are no available write request credits
- e. Write requests issued to network to a specified destination node
- f. Write requests above a specified size
- g. Latency measurements (counting clocks from when a request is issued until the last beat of the response is returned)
- h. Measure the minimum/maximum pulse width of a filtered event – this could be used to measure the minimum and maximum latency of a request/response pair
  - ♦ Link events
- i. Requests (flits) through the link
- j. Total number of beats (phits) through the link
- k. Backpressure on a link when there is data to transfer on the link
- l. Requests that have the firewall error bit set

## 19.5 PMON interface to Trace/Debug

PMON events can be used for triggers and can be exported to a trace bus. Any PMON event (both raw and filtered) can be used as a trigger and/or exported. The debug document talks about a master trigger generation block. PMON events can be provisioned to feed into a master trigger generation block, so PMON events can be used in generating sophisticated triggers. For example, suppose that every time a write occurs to a particular destination, another write to that destination should not occur before a response to the first write is received. A trigger can be setup in a master trigger generation block using filtered PMON events for write and write responses and the resources of the master trigger generation block.

## 19.6 PMON Interrupt Capability

PMON events can be used for interrupts. Any PMON event (both raw and filtered) can be defined to be an interrupt.

## 19.7 PMON Access

There are four ways that PMON interacts with the system external to the interconnect:

1. PMON control and status registers are accessed through the configuration ATU(s). Note that PMON components that are dedicated to a given VN may only be accessed by a configuration ATU that has access to that VN's configuration space and sufficient privilege level.

2. PMON interrupts – PMON interrupts feed into the interrupt structure described in the document that details Symphony's interrupt capabilities.
3. Debug/Trace - PMON events can be used for triggers and can be bundled into a trace sample
4. Imported Events - External Events can be brought from outside the interconnect and be used as triggers, events or bundled into a trace.



# 20

# Debug and Trace

This section defines the debug/trace architecture in Symphony. Note that the performance monitoring and debug/trace functionality (described in Section Performance Monitoring (PMON) on page 287) have common concepts and that elements of one can feed into the functionality of the other.

## 20.1 Definition of Debug

The following attributes define debug in the context of Symphony:

1. Access to internal state information that can be used to debug issues relating to the Symphony system. Examples of internal state are:
  - a. State of building blocks internal to the Symphony system:
    - ◆ Switch
    - ◆ ATU
    - ◆ Buffer
    - ◆ Firewall
    - ◆ Other building blocks not listed here
  - b. Value of flow control signals between elements of the system such as:
    - ◆ Valid/Ready flow control signals that control packet flow on an internal link
    - ◆ Grant signals on a link
    - ◆ Flow control signals on external interfaces
  - c. Statistical information or conditioned information:
    - ◆ Counts of identified events (such as write requests, etc.)
    - ◆ Timer values – count of clock ticks since an event counter is enabled. The timer value can be used as a time reference for calculating latencies between two events.
2. How debug information is accessed:
  - a. Through a dedicated external debug (trace) interface, such as:
    - ◆ AMBA Trace Bus (ATB)
    - ◆ MIPI STPv2 with encapsulated ATB packets.
  - b. Through “debug” registers in Symphony’s configuration address space that provide a snapshot of some defined state. This may include a configuration backdoor

- access of state stored in select on-chip memory or registers (like context memory in the ATU, for example).
- c. Through performance counters. They contain a form of aggregated state based on what is selected to count.
  3. How debug information is obtained:
    - a. Assembling what gets put on the trace bus
    - b. Creating trigger start and stop events

## 20.2 Symphony Debug Elements

The debug function in Symphony is created based on a group of building blocks/components described below:

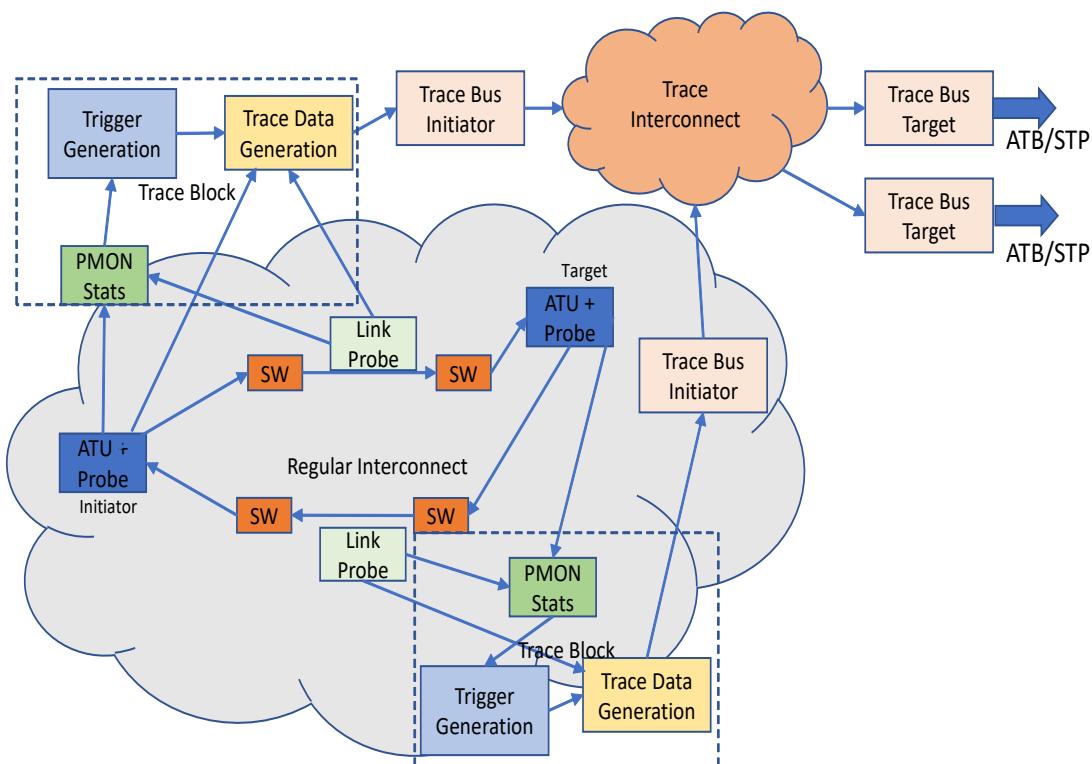
1. Probes – a probe is an observation point internal to Symphony. Probes are chosen by the user at design time. Probe points can be sourced from the following:
  - a. Any PMON event or associated trace data. For example, a portion of the header data associated with a header flit could be put on a probe that is used to generate a trace sample.
  - b. Selected state from blocks inside a trace block:
    - ◆ PMON statistics blocks, such as timer values or events that can be generated by the statistics blocks
    - ◆ Trigger generation blocks (described later in this document)
2. Trace block – this component takes inputs from probes and performs some type of processing on them. This block in turn consists of all or some of the following sub-blocks:
  - a. PMON statistics block
  - b. Trigger generation block
  - c. Trace data generation block

Users instantiate these components at design time and define what probes are connected to a given trace block. This block contains configuration registers, since the blocks inside are programmable and/or have processing results.
3. PMON statistics block - this component is just as described in the PMON chapter. As part of debug infrastructure it receives events from probes which are used to generate other events based on counter and timer values. The outputs of this block feed the inputs of trigger generation block as well as trace data generation block.
4. Trigger generation block - this component is described in section 20.5.
5. Trace data generation block - This receives trace data from probes and performs some filtering according to specified criteria. The filtered trace data is fed to the Trace bus initiator.
6. Trace bus initiator – this component takes trace data and triggers from a trace block and creates a trace beat – generating a conditional sample that will ultimately end up on an external trace bus. Think of this as the initiator ATU of the trace world. It acts as a source on a “trace interconnect.”

7. Trace interconnect – a network (or networks) that sits between one or more trace bus initiators and one or more trace bus targets. It acts as the transport for trace bus data from trace bus sources in Symphony to the trace bus interfaces of Symphony.
8. Trace bus target – this component takes trace beats received from the trace interconnect and drives the external trace bus interface with them. This is the target ATU of the trace world. In addition, this component can also optionally have a “trace processor” function. This module is a central point for trace data to flow through, and as such the trace processor could perform things such as the following:
  - a. latency calculation based on timestamps on start and stop traces
  - b. latency histograms
  - c. BW calculations
9. The point here is not to define what type of processing is done, but that processing can be done. Access to the results would either be through the trace bus or configuration registers.

A block diagram showing the debug/trace elements in a Symphony interconnect is shown in Figure 20-1 on page 295.

**Figure 20-1 Debug Elements in a Symphony Interconnect**



## 20.3 External trace interfaces

Symphony will support two industry standard trace interfaces:

1. ATB
2. MIPI STPv2 with encapsulated ATB packets

Symphony can be provisioned to have one or more trace interfaces or no trace interfaces. The following attributes can be assigned to each trace interface:

1. Trace interface type (ATB or STP)
2. Trace interface width

As described earlier, an external trace interface is driven by a trace bus target.

## 20.4 Symphony will not Support Large Explicit Internal Trace Buffers

The trace and debug ecosystem has canned trace buffers that can be instantiated as needed by the customer external to the Symphony subsystem. However, buffers of limited size can be put in the Trace Bus Initiator, Trace Interconnect, and Trace Bus Target. These effectively act as trace buffers.

## 20.5 Triggers

Symphony will have sophisticated trigger generation capability. Requirements for trigger generation include:

1. Logic analyzer-like trigger generation:
  - a. Capability to implement a simple state machine to generate a trigger. The idea is that we will have a “trigger generation component” that a customer can instantiate in Symphony.
  - b. State machine will have a limited number of states – 8 to 16
  - c. State machine will support a limited number of if-else clauses – 4 to 8
  - d. The sequencing of the state machine will be controlled by configuration registers – in essence, the configuration registers hold the micro-code of the state machine.
  - e. Any probe point can be fed into the trigger generation logic. Events from performance components can be used as inputs to the state machine. Counters can also be provisioned to impact state machine flow. For example, events and/or clock ticks could be counted, and the control of the counters can be performed from the state machine (like a LA). Note that there will be a concept of “performance components” – these components can be provisioned by the customer into Symphony at desired locations. These components can be provisioned to generate trigger events based on values of internal performance counters. See the document on performance monitoring for more details on this.

2. Some examples of trigger generation based on this scheme:
  - a. Trigger based on a performance counter exceeding a threshold
  - b. Event arms trigger; if a subsequent defined event does not happen within a certain count, trigger
  - c. Trigger based on how many times a firewall fires (this implies that the firewall can generate an event that inputs into the event connection “fabric”).

## 20.6 Trace Bus Formation

The external trace data bus output will be composed of probes that have been connected to trace bus initiators. The customer will be able to pick which probes are placed on the trace bus and how they are arranged. If there are multiple external trace busses, then the customer can route from any trace bus initiator to any trace bus target through the Trace interconnect. The external trace bus includes an ID field as part of the output, so if there are multiple trace sources (trace bus initiators), each one will have a unique ID and that ID will show up on the external trace bus output. Here are three mechanisms on how to provide the IDs for each trace bus source – we will support all three:

1. Input bus per trace source that is tied to the desired value by the customer
2. Parameter per trace source that is set by the customer
3. Configuration register per trace source that is programmed by the customer

## 20.7 External Timestamp

An external timestamp shall be provided to Symphony and used by the debug logic. There is no plan to provide an internally generated timestamp.

Care should be taken when using the external timestamp to guarantee all events that can be stamped by a timestamp and occur at the same time will record a timestamp value as close to each other as possible given the constraints of pipelining for timing closure and synchronization.

## 20.8 External trigger outputs

Symphony will export output triggers as provisioned by the customer. Any internal trigger can be provisioned to be an output trigger.

## 20.9 External trigger or events inputs

Symphony will import triggers or events as provisions by the customer. Events can become part of a trace output.

## 20.10 CoreSight Compatibility

Symphony will provide some CoreSight Compatibility and will not be CoreSight compliant. This means Symphony will include the implementation of

1. A set of registers that provide a CoreSight Programmer's model

Symphony, however, will not support Topology detection and registers for auto-discovery.

Symphony shall provide ATB and STP interfaces as mentioned in Section 20.3. This makes Symphony debug/trace architecture partially CoreSight Reusable.

The functionality of the External triggers should be configurable and done in such a manner as to provide the functionality needed by the CoreSight signals TRIGIN, TRIGINACK, TRIGOUT and TRIGOUTACK.

## 20.11 Configuration

Configuration related to debug (trace control and generation as well as performance monitoring blocks that feed into trace) will be required. The requirements for configuration are:

1. A special JTAG interface is not required. ARM deals with controlling debug from JTAG by muxing in JTAG control upstream from the debug configuration interface(s) that feed into Symphony. The next requirement provides additional background on this.
2. The debug configuration interface should be able to either be shared with the configuration register interface used for provisioning other aspects of Symphony or be a dedicated configuration interface. ARM has system diagrams where the configuration bus used for trace control is dedicated to debug, so anything we do on the configuration interface should not preclude the capability to do this.
3. Debug-related registers that service different virtual networks may be required to be on different configuration busses. This is needed if a customer has a requirement that debug-related control and status on one virtual network (VN) is separate from other VNs. This requirement may lead to there being more than one configuration interface for debug-related register accesses.
4. Customers can require that accessing debug-related registers be done securely. This requirement needs to be supported by our configuration architecture.

# 21

# Routing

## 21.1 Objective

The objective of routing is to determine a path from the source to the destination through the Interconnect such that: 1) the QoS requirements of the flow are met; 2) overall routing of flows maximizes the carried traffic and minimizes the overall cost of the Interconnect; and 3) does not cause deadlocks.

Symphony will primarily support oblivious source routing but also has the capability to support incremental routing.

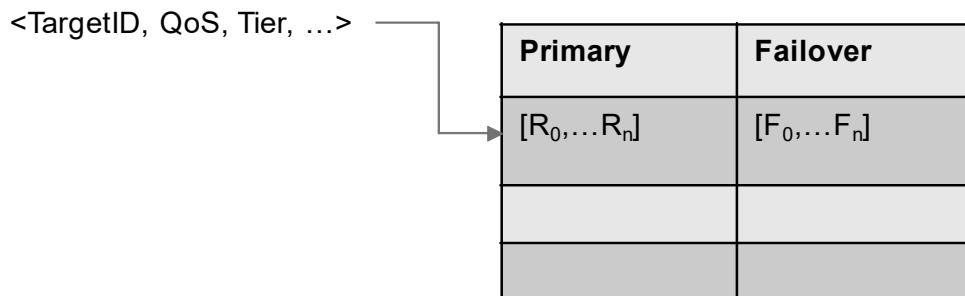
## 21.2 Oblivious Source Routing

Symphony will support deterministic source routing. In deterministic source routing as opposed to adaptive routing.

In Source Routing the complete path that a packet takes to reach its destination is precomputed and loaded in the table at the Source I-ATU. The CTL layer in Presto and the Route Look Up function in Legato is responsible for performing the route look up function.

With source routing, the selected route is prepended to the packet header. The look table may contain more than one route for the flow. In which case, an appropriate route will be used based on specific system policy. For example, a look up can return a primary route and a backup route in case the system detects that the primary route is not functional any more.

The look up is not limited to only Target ID but can include the following tuple along with the Target ID: <Target ID, QoS, Steer/Tier, ...>, such that for the same Target ID a different route can be picked based on QoS/Steer/Tier values. Routing can be done based on other fields within the packet header as well.



**Figure 21-1 Source Routing Table**

The look up will return a route vector [R<sub>0</sub>, ..., R<sub>n</sub>] where R<sub>i</sub> are the bits that switch "i" will use to route the packets.

Each switch in the path will use the relevant bits to choose the output port and then shift the route bits appropriately so that the next switch can look at the next set of bits starting the LSB. Each switch knows how many bits it should look at, depending on the number of outgoing ports of the switch.

Route bits will be parity protected. Incremental Source Routing

## 21.3 Incremental Routing

With Incremental routing, the complete path of the packet from source to destination is not available at the source I-ATU. Rather each node in the network only forwards the packet one-hop to the next switch. Each node in the network has a routing table which is looked up. Based on the <TargetID, QoS, Tier, ...> tuple the lookup returns the egress port of the current switch.

## 21.4 Dynamic Routing

Symphony will not support Dynamic Routing. Differentiation based on packet properties

Certain packet properties as reflected by the QoS and Tier bits can be used to determine the VC on the outgoing port that the packet should use.

This property allows the network to split the VCs as well as merge them where necessary.

VC allocation of packets per QoS/Tier or other attributes is predetermined by Symphony software. This predetermination is done to meet the overall power, performance and area goals of the Interconnect.

---

**Implementation**  
This enables the ability to create separate networks to carry reads vs. writes, snoops vs. non-snoops or to take alternate route when certain conditions are met.

Practically speaking, while any information in the packet can be used, such as write data, the data closest to the front will be what is used. Additionally, the number of bits used to extend the Target ID will be limited because of timing concerns. Because of these concerns, micro-architecture may limit the number of bits and which fields in the header that can be used.

---

## 21.5 Routing In Symphony

This section explains how packets would be routed in Symphony.

The Symphony packet definition contains the following fields that can be used by the Routing algorithm to determine the complete path of the packet through the network.

**Table 21-1 Packet fields in the header used to route the packet**

Field	Routing Usage
Routing Field/ Destination ID	Determines the Egress port for the packet on the Network Element

QoS/Pri Field	Determines which VC (if implemented) on the Egress Port the packet should be mapped to. How the packet is scheduled for transmission is implicitly determined by the VC it is mapped to.
Tier	Determines which VC (if implemented) on the Egress Port the packet should be mapped to.
Virtual Network	Determines which VC (if implemented) on the Egress Port the packet should be mapped to.

## 21.5.1 Mapping of Flows and Virtual Paths

Definitions:

**Flow:** A flow uniquely defines a set of messages that are defined by the tuple {SrcID, DestID, QoS, VN, address}. Flows are unidirectional and are defined to traverse from the source to destination in the Interconnect.

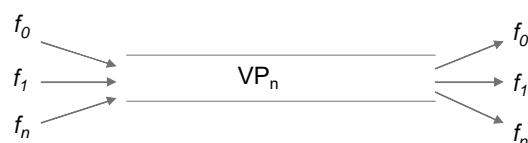
It should be noted that a request flow may also request a response flow. However both these flows, although related are treated as two independent flows. **Virtual Path:** A path is a distinct sequence of Fabric components (ATUs, switches, links and adapters) and has the following properties: 1) Has a beginning and an end; 2) Each component in the sequence is adjacent to another; 3) No link is repeated in the path or if VCs are present no VC is repeated; and 4) path is unidirectional.

Flows are uniquely defined on a end-to-end basis and may go across multiple fabrics. The Virtual Paths (VP) on the other hand have relevance within a Fabric only.

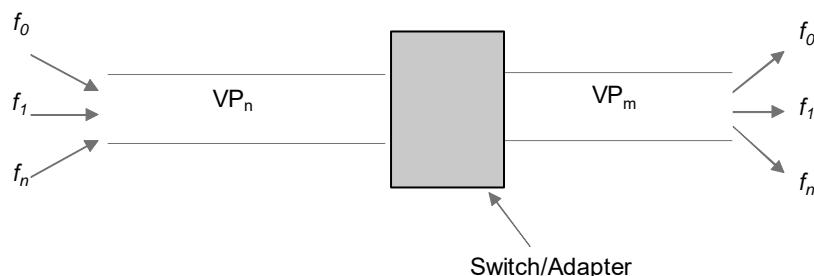
Flows do not carry any routing information or how the packets will be carried through the network. The flow table that will be constructed by SW will contain information about the bandwidth required, the latency and delay jitter requirements of the flow along with protection and security attributes. [see definitions at the beginning of the document]

Virtual Paths can be thought of as containers, and contain routing information, the QoS value as determined by the system (mapping of Flow QoS to VP QoS), virtual network mapping, resiliency and protection attributes. Multiple flows can map into a single Virtual Path. Virtual Paths can be concatenated, aggregated or split as needed.

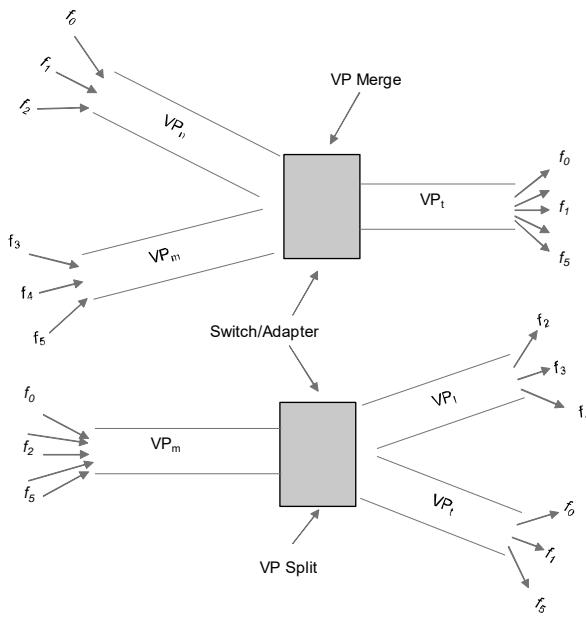
**Figure 21-2 Multiple flows can be mapped to the same VP**



**Figure 21-3 VP Concatenation**



**Figure 21-4 VP merging and splitting**



VPs are represented as follows:

$VP_i = [\{Flow\ Map\}, \{Route\}, \{QoS\}, \{Priority/Tier\}, \{SrcID\}, \{DestID\}, \{VN\}, \{Protection\}, \{Security\}]$

Where:

$\{Route\} = [[NE_i, EgressPort_j, VC_k], \dots, \dots, \{NE_t, EgressPort_u, VC_t\}]$ , where first element is the Src NE for the VP and the last element is the Dest NE for the VP.

————— Note —————  
Route for individual flows is given in the following format.  $\{Route\ (flow)\} = [[NE_i, EgressPort_j, VC_k], \dots, \dots, \{NE_t, EgressPort_u, VC_t\}]$ , where  $((), (), ..)$  is an ordered list of switches and elements the flow is mapped on.

VPs will generally be used to group flows with similar QoS requirements. The architecture, however, allows for different types of flows to be grouped into a single VP.

{Flow Map}:

- ▶ Map of all the flows that are part of the VP.
- ▶ Syntax:  $[\{Flow_0, Flow_1, \dots, Flow_n\}]$

Let  $f_x(t_0, t_1, \dots, t_n) \rightarrow t$ , then:

{QoS}:

- ▶ QoS value that is associated with the VP is:  $f_{qos}(QoS_0, QoS_1, \dots, QoS_n) \rightarrow QoS_{vp}$ , where  $f_{qos}$  is a function determined by SW.

Similarly,

{Security}:

- ▶ Security value that is associated with the VP is:  $f_{Sec}(Sec_0, Sec_1, \dots, Sec_n) \rightarrow Sec_{vp}$  where  $f_{Sec}$  is a function determined by SW.

{Protection}:

- ▶ Protection associated with the VP is:  $f_{Prot}(Prot_0, Prot_1, \dots, Prot_n) \rightarrow Prot_{VP}$  where  $f_{Prot}$  is a function determined by SW.

{SrcID}:

- ▶ The ID of the NE that the VP originates.

{DestID}

- ▶ The ID of the NE where the VP terminates.

It should be noted that as the flows goes through VPs, its packet header does not change. What changes is how the QoS/Tier/Prot/Security fields are interpreted and mapped. The VP provides a hint of how the mapping is done.

If we define  $R(n) \rightarrow \{\text{Self Routing bits}\}$  where  $R(n)$  is the function that takes the route “n” and outputs the self routing bits for the packet header, then we will have the following:

$$R(r_{vp1}, r_{vp2}, r_{vp3}, \dots, r_{vpn}) \rightarrow \{\text{Self Routing Bits}\}$$

Where the flow is part of  $VP_i$  for all  $i$  that the flow is mapped to. Similarly, the mapping from flow QoS to NE arbitration scheme, arbitration weights, and VC is defined by the QoS value of the VP.

The “Self Routing Bits” determine the egress port on the NE. The “QoS/Tier -to-VC” mapping in the NE determines which VC on the egress port the packet will exit.

---

————— Note —————

VC allocation can change from one NE to another in the network. It would be too restrictive in the network to expect that if a flow was mapped to a particular VC number at beginning of Fabric, that it would maintain the VC numbering across the rest of the Fabric.

---

## 21.5.2 Flow Map

Flow map will be constructed by Maestro and will consist of the following information:

**Table 21-2 Flow Map**

Flow #n	Initiator	Target	Traffic Profile	Delay	Jitter	Eff BW	Pri	Flow Type	Credit	VN ID	Sec	Fail Op
1	RT_CC	Mem	<ul style="list-style-type: none"> <li>Peak = 100 MB</li> <li>Burst = 250B</li> <li>Avg = 50MB</li> </ul>	100ns	125ns	75MB	P0	RT	Yes	0	NS=1	<ul style="list-style-type: none"> <li>Fit rate = "x"</li> <li>Route Dup = Yes</li> <li>ECC/Parity = ECC</li> </ul>
2	GE_3	Mem	<ul style="list-style-type: none"> <li>Peak = ..</li> <li>Burst = 512B</li> <li>Avg = 10MB</li> </ul>	125ns	150ns	10MB	P0	RT	Yes	0	NS=0	<ul style="list-style-type: none"> <li>Fit rate = "x"</li> <li>Route Dup = Yes</li> <li>ECC/Parity = ECC</li> </ul>
3	CC_1	Mem	<ul style="list-style-type: none"> <li>Peak = 200MB</li> <li>Burst = 128B</li> <li>Avg = 100MB</li> </ul>	200 ns	225ns	125MB	P1	LL	Yes	1	NS=0	<ul style="list-style-type: none"> <li>Fit rate = "y"</li> <li>Route Dup = No</li> <li>ECC/Parity = Parity</li> </ul>
...	....	...	• ....	...	...	...	...	...	....	...	...	

Some not so obvious terms are defined below:

- ▶ Eff BW: Effective Bandwidth

- ▶ Effective bandwidth is defined as the bandwidth that is actually allocated to the flow. A flow may be defined by the peak bandwidth that it needs or by the average bandwidth or by a combination of peak and average bandwidth. Effective bandwidth is a bandwidth that is allocated based either on the peak, average or a combination of both.

► Credit:

- ♦ The credit field indicates whether the flow is controlled by the credit mechanism. Not all flows need to be controlled by using credits.

► Fail Op: Fail Operational

- ♦ Fail operational attribute of the flow may include whether resource duplication is required in NEs to which the flow is mapped on and type of protection needed for the flow.

### 21.5.3 Virtual Path Map

The VP Map has the following attributes:

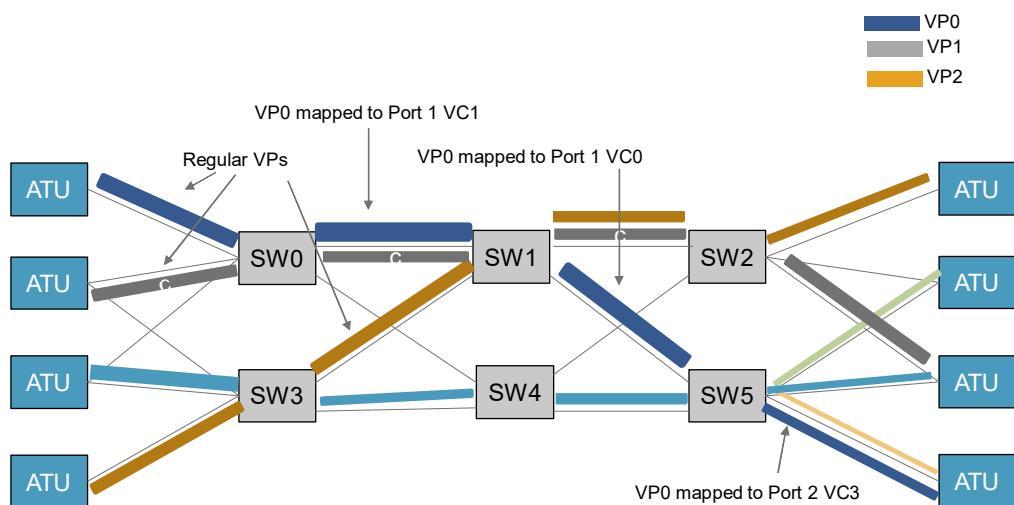
- Bandwidth that needs to be reserved between  $\text{SrcID}_{vp}$  and  $\text{DestID}_{vp}$ .
- Route.
- QoS map, i.e., how will the different NEs along the path map flow QoSs in to VCs.
- Protection and resiliency mechanisms need to be employed along the route.
- Multiple VPs can be mapped to the same VC on the NE.

**Table 21-3 VP Map**

VP #	SrcID	DestID	Flow Map	Traffic Profile	QoS	Eff BW	Route	VP Type	VN ID	Sec	Fail Op
1	NE ID_3	NE ID_5	• F1, F2, F5, F6	• Peak = 100 MB • Burst = 256B • Avg = 50MB	• P0	75MB	.....	RT	0	NS=1	• Fit rate = "x" • Route Dup = Yes • ECC/Parity = ECC

An example of the VPs is illustrated below in Figure x. In this example, VP0 is mapped to Port 1 and VC1 on switch SW0, to Port 1 VC0 on switch SW1 and to Port 2 VC3 on switch SW5.

**Figure 21-5 VP mapping showing a single VP mapped to different VCs along the path**



---

Note

---

Same VP can be mapped to different VCs at different switches in the network.

---

## 21.5.4 Example

The example below, illustrates how two flows, Flow 1 and Flow 2, traverse through a series of switches in the network illustrated below. The example is meant to illustrate how the flows map to VCs as they traverse the network and how VPs work. Note: as mentioned above, VPs act as containers. Every flow that is mapped to the VP container, is treated in the same way. VP containers can be merged or split as shown in the Figure below, where VP0 and VP1 merge to form VP3 (traversing SW4 and SW6). VP2 is then split up into VP3 carrying flow 2 and VP4 carrying flow 1.

In the figure below, flow 1 is mapped to different VCs along the path. This mapping could be based on how the QoS values are mapped on the egress port and which VC best represents that QoS value.

In the example below, the route bits identify the egress ports at the switches that the packet would take to get to the destination. In the example below, Flow 1 and Flow 2 will carry the following routing and QoS information in their header:

- ▶  $R_{Flow1}: \{10011\}$ ;  $QoS_{Flow1}: \{0001\}$
- ▶  $R_{Flow2}: \{01000\}$ ;  $QoS_{Flow2}: \{0011\}$

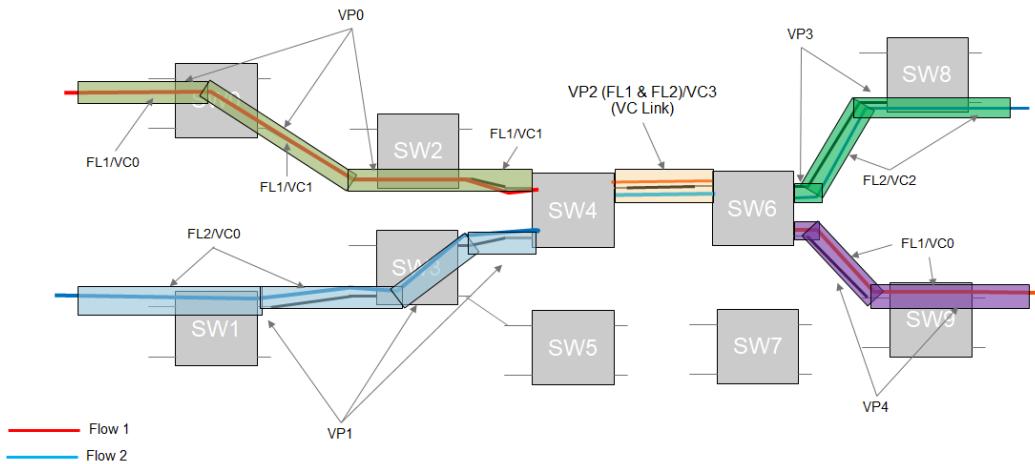
The VC selection is done in the example below on the QoS values. The QoS mappings each switch and flow are as follows:

**Table 21-4 QoS-VC mapping**

Flow QoS	SW0	SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8	SW9
Flow 1 (0001)	VC1	NA	VC1	NA	VC3	NA	VC2	NA	VC2	NA
Flow 2 (0011)	NA	VC0	NA	VC0	VC3	NA	VC0	NA	NA	VC0

VP merging happens when VP0 and VP1 are merged on to VP2. VP splitting happens when VP2 is split into VP3 and VP4.

In nutshell, a flow can be mapped to multiple VPs and multiple VCs as the flow traverses the network.

**Figure 21-6 Relationship between VPs, VCs and Flows**

## 21.6 Deadlocks

Deadlocks occur when packets worm-holing through the Fabric are held up waiting for resources (buffers/channels) to free up in such a manner that dependency is circular.

When a deadlock occurs, packets stop to make forward progress. One way to avoid deadlocks is to make sure that there is no circular dependency between packets seeking resources to make forward progress.

One of the ways to determine if there is a deadlock, is to construct a “Resource Dependency Graph (RDG)”.

## 21.7 Resource Dependency Graph (RDG)

Resource dependency graph is directed graph of the resources that are expected to be used by different flows in the Fabric/Interconnect. The RDG is defined as  $D = G(C, E)$  where  $C$  denotes the  $[Switch_i, Port_j, VC_k]$  for all “ $i$ ” an element of the set of switches,  $j$  an element of the set of output ports of switch  $i$ , and  $k$  an element of the set of VCs on output port  $j$  that a flow is mapped to, and  $E$  as the edges that connect the nodes in  $C$  together.

In Symphony, the resource dependency graph needs to be constructed for each flow in the system.

A sufficient condition for deadlock free routing is that there be no cycles in the channel dependency graph. That is a deadlock is created by a set of flows such that those flows request resources that are also requested by other flows in such a manner that there is dependency cycle between the resources requested by these flows.

---

Note

---

*Dally's Theorem 1 for Deterministic Routing: The routing function  $R$  is deadlock free iff there are no cycles in the Resource Dependency Graph [W. Dally and C. Seitz, “Deadlock Free Message Routing in multiprocessor Interconnection Networks”, CS dept, Caltech TR 86]*

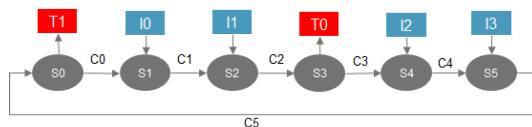
---

One way to prevent deadlocks would be to discover all Strongly Connected Components in the graph and remove them without affecting the connectivity of the graph.

Tarjan's algorithm detects strongly connected components in a directed graph in  $O(C+E)$  time. Tarjan's algorithm uses the DFS algorithm to detect SCC.

If the algorithm finds a cycle, then the cycle has to be removed by adding an escape route. This escape route can be a different VC over which one of the flows causing the cycle is routed or an additional physical link.

Consider the ring network show in the figure below.



**Figure 21-7 Ring Network. I0 & I1 transmit to T1 and I2 & I3 transmit to T0.**

In the figure below, the RDG is constructed using the channels as the vertex and edges as the dependency between the channels. Notice that there is a cycle in the RDG.

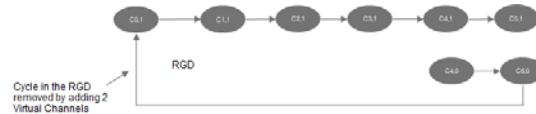


**Figure 21-8 RDG for the Ring Network in Figure 21-7.**

Once a cycle is discovered by using Tarjan's algorithm, it can be removed easily by adding two Virtual Channels. Let the rule to choose a particular virtual channel be as follows:

- ▶ Use VC0 if current switch (number) < destination switch.
- ▶ Use VC1 if the current switch (number) > destination switch.

Based on the above rule, the RDG is constructed again as shown in the Figure below.



**Figure 21-9 RDG when 2 VCs are added to the network.**

Notice that not all switches need to support two VCs. In the above case only switches 4 & 5 need to support 2 VCs. The rest of the switches can all be single VC switches. The reason why this is important, is because adding VCs adds to the area of the switch.

## 21.8 Networks and Deadlocks

Interconnect networks can be subdivided into the following categories:

- ▶ Regular networks and Irregular networks.
- ▶ Direct and indirect networks.

A regular network topology is defined in terms of a regular graph structure (such as a mesh, ring, torus, etc.). In a regular topology, each node has the same numbers of neighbors, and hence regular topologies

are extensible by simply adding stages to the previous structure. Since regular topologies are based on a regular graph structure, routing is relatively easier in such structures.

Irregular topologies do not have a regular graph structure. The Symphony architecture will support Endpoint interconnect architecture referred to in the literature as an indirect Interconnect. However, the architecture will not preclude a direct interconnect structure as well.

Direct interconnects are those for which the endpoints sit inside the Interconnect. Indirect interconnects are those for which the endpoints sit outside the Interconnect.

Topologies such as Torus and Mesh lend themselves naturally to a direct interconnect structure, while others such as Benes multistage networks, ShuffleNets etc. are more affine to indirect/Endpoint interconnects.

Indirect regular networks such as Benes multistage networks or ShuffleNets or a Crossbar with separate request and response network do not have deadlock issues as there are no cycles in the RDG. Benes and Shufflenet Networks may be blocking depending on the dilation in the middle stages but do not deadlock. Indirect regular networks with same network carrying both request and response traffic but on different VCs should be deadlock free as well.

Indirect irregular networks can result in deadlocks and hence an RDG needs to be created and examined before declaring that there are not deadlocks in the network.

Direct networks can result in deadlocks whether they are built out of regular network structures or irregular structures.

**Table 21-5 Type of Network and potential deadlock**

Type of Network	Network Structure	Potential Deadlock
Indirect	Regular	<ul style="list-style-type: none"> <li>- No.</li> <li>- No need to develop the RDG and look for cycles. Creating an RDG and looking for cycles is time consuming and in the best case is of linear order and worst polynomial.</li> </ul> <p><a href="#">Examples: Benes, Shufflenet, Tree</a></p>
Indirect	Irregular	<ul style="list-style-type: none"> <li>- Yes.</li> <li>- Maestro Software should create an RDG to determine and eliminate any potential cycles and deadlocks.</li> </ul>
Direct	Regular or Irregular	<ul style="list-style-type: none"> <li>- Yes.</li> <li>- Maestro Software should create an RDG to determine and eliminate any potential cycles and deadlocks.</li> </ul>

## 21.9 Deadlock Avoidance Algorithm:

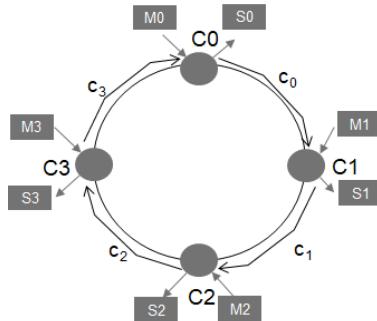
The deadlock avoidance algorithm is described as follows:

1. Construct a Flow Map.
2. Construct a Resource Dependency Graph (RDG).
3. Search for cycles in the RDG.
4. Remove the cycles in the RDG by either rerouting the flow/flows or adding VCs.
5. Go back to step 2.

## 21.10 Deadlock Example

Let us take a simple example of unidirectional ring network as shown in the Figure below.

**Figure 21-10 Simple unidirectional ring network with 4 initiators and 4 targets and no-VCs.**



In the figure above, there are 4 Masters accompanied by 4 Slaves respectively. Each Master and Slave pair is connected to a Switch “CX”. The link that connects two switches is named “cx” where “x” is the switch the link originates from.

Let the flow between master “MX” and Slave “SY” be defined as “ $f_{xy}$ ”. The flow map is shown in Table below:

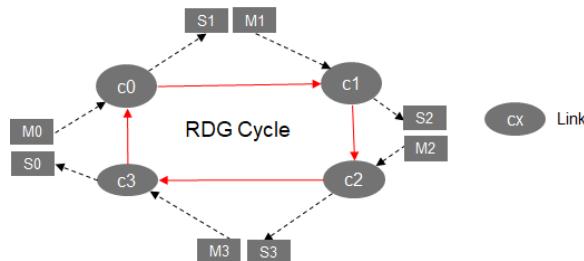
**Table 21-6 Flow Map and Route**

Src->Dest	Flow Name	Route
M0->S1,S2,S3	$f_{01}, f_{02}, f_{03}$	{c <sub>0</sub> }, {c <sub>0</sub> , c <sub>1</sub> }, {c <sub>0</sub> , c <sub>1</sub> , c <sub>2</sub> }
M1->S2,S3,S0	$f_{10}, f_{12}, f_{13}$	{c <sub>1</sub> }, {c <sub>1</sub> , c <sub>2</sub> }, {c <sub>1</sub> , c <sub>2</sub> , c <sub>3</sub> }
M2->S3,S0,S1	$f_{23}, f_{20}, f_{21}$	{c <sub>2</sub> }, {c <sub>2</sub> , c <sub>3</sub> }, {c <sub>2</sub> , c <sub>3</sub> , c <sub>0</sub> }
M3->S0,S1,S2	$f_{23}, f_{20}, f_{21}$	{c <sub>3</sub> }, {c <sub>3</sub> , c <sub>0</sub> }, {c <sub>3</sub> , c <sub>0</sub> , c <sub>1</sub> }

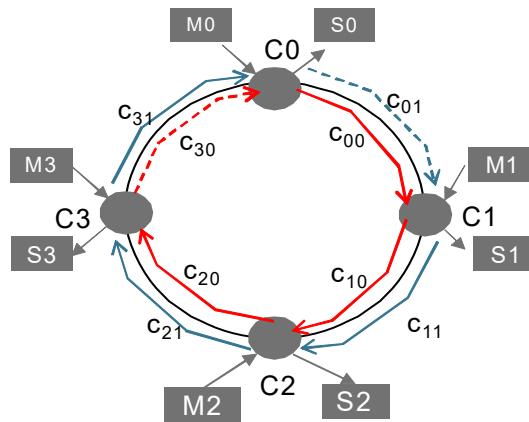
Based on the above flow map, we construct the RDG for the network. Figure below shows the RDG. Notice that there is a cycle in the RDG. This cycle has to be removed.

The cycle created is removed by adding Virtual Channels.

**Figure 21-11 RDG for the Network shown above**



In the Figure below, each switch has two Virtual Channels: VC0 and VC1, denoted by “ $cxy$ ” where “x” is the originating switch and “y” is the virtual channel number.

**Figure 21-12 Unidirectional Ring with 4 Masters and 4 Slaves each supporting 2VCs.**

The table below captures the flow map for the above network. The routing algorithm we used is as follows. Algo:

**Table 21-7 Flow Table for Network Shown Above**

<b>Src -&gt; Dest</b>	<b>FFlow Name</b>	<b>Route</b>
M0->S1,S2,S3	f <sub>01</sub> , f <sub>02</sub> , f <sub>03</sub>	{c <sub>00</sub> }, {c <sub>00</sub> , c <sub>10</sub> }, {c <sub>00</sub> , c <sub>10</sub> , c <sub>20</sub> }
M1->S2,S3,S0	f <sub>10</sub> , f <sub>12</sub> , f <sub>13</sub>	{c <sub>10</sub> }, {c <sub>10</sub> , c <sub>20</sub> }, {c <sub>11</sub> , c <sub>21</sub> , c <sub>31</sub> }
M2->S3,S0,S1	f <sub>23</sub> , f <sub>20</sub> , f <sub>21</sub>	{c <sub>20</sub> }, {c <sub>21</sub> , c <sub>31</sub> }, {c <sub>21</sub> , c <sub>31</sub> , c <sub>00</sub> }
M3->S0,S1,S2	f <sub>23</sub> , f <sub>20</sub> , f <sub>21</sub>	{c <sub>31</sub> }, {c <sub>31</sub> , c <sub>00</sub> }, {c <sub>31</sub> , c <sub>00</sub> , c <sub>10</sub> }

1. If the current node ID <= DestID use VC0
2. Else use VC1.

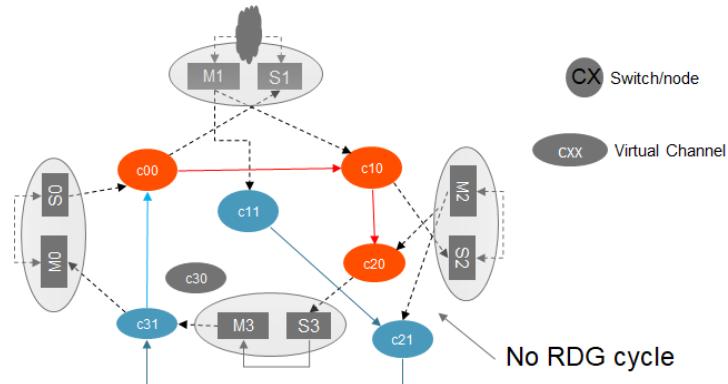
---

**Note**

The above algorithm of assigning VCs will be used for (unidirectional) Meshes and other regular structures.

---

Based on the above Flow Table, the Figure below shows the corresponding RDG. Notice that the RDG has no cycles in it! Red arrows represent VC0 and blue arrows represent VC1.

**Figure 21-13 RDG for the Network shown above.**

## 21.11 Detecting System Level Deadlocks

The Flow RDG can detect deadlocks within the flow context, but it would not detect deadlocks caused by wrong system configuration. The system may include multiple Symphony Fabrics and/or customer proprietary Fabrics.

User only knows the socket level dependency. Maestro/Symphony would not know the dependency.

We define “Connection” as follows:

“A connection is composed of one or more flows and has end-to-end semantics. Connections are unidirectional. Connection “C” is formally defined as follows:

$$C \equiv \{(M_i \rightarrow S_j), (M_t \rightarrow S_k), \dots\} \forall (i \in \text{the connection spans}) \text{ where } M \text{ is the Master and } S \text{ is the Slave.}$$

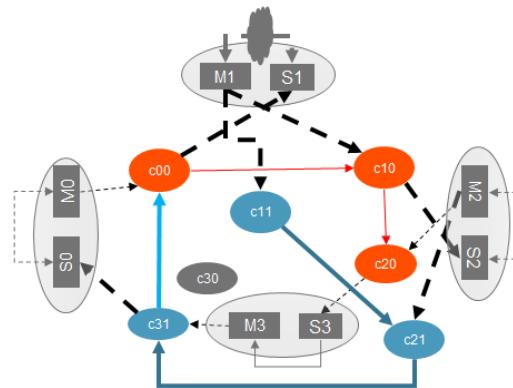
such that flows  $\{f_i, f_t, \dots\}$  connect “M” to their respective “S”.

With the above formulation we can state the following rules:

- a. If any of the “M” or “S” are repeated in the connection list, dependencies need to be investigated such as reuse of AXI\_ID, or reassembly buffers or ordering buffers.
- b. If the flows intersect or converge, dependencies need to be investigated.

Consider the network shown in Figure 21-13 above and consider connection “C1” defined as follows:  $C1 := \{M1 \rightarrow S2; M2 \rightarrow S1; M1 \rightarrow S0\}$  using flows  $\{f_{12}, f_{21}, f_{10}\}$ . The figure below illustrates the connection and the flows.

**Figure 21-14 System level RDG for Connection C1.**

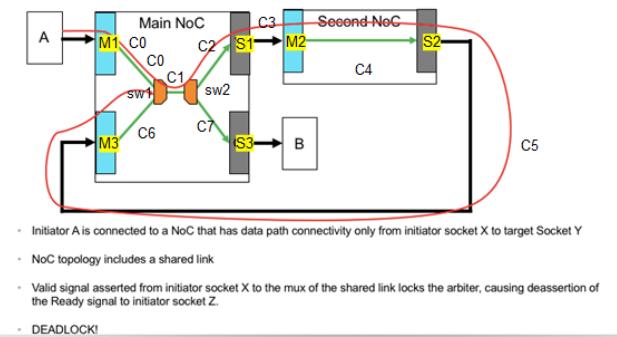


Notice that “M1” is repeated in the connection list and the flows  $f_{21}$  and  $f_{10}$  intersect. Checks need to be made if the AXI ID is being reused at M1 by flow  $f_{21}$ , or if there is a conflict for buffer space as would be the case for re-ordering or splitting of transactions.

Rules for Reorder Buffer, I-ATUs, T-ATU, Splitting of transactions, etc. will be put in place in due course of time.

Another example of system level is shown in Figure 21-15 below. In Figure 21-15, transaction is sent from Master A and destined for Slave B.

### Roundabout deadlock example



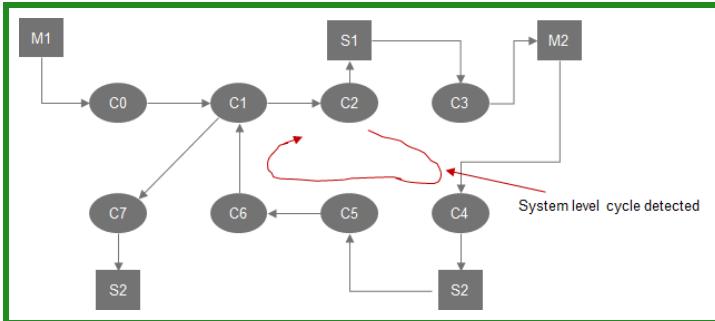
**Figure 21-15 System level deadlock example.**

The flow  $f_{ab}$  is represented as follows:

$$f_{AB} = \{\{M1, C0, SW1, C1, SW2, C2, S1\}, \{S1, C3, M2, C4, S2\}, \{S2, C5, M3, C6, SW1, C1, SW2, C7, S3\}\}$$

The system level RDG is represented in below.

**Figure 21-16 System level RDG for the network shown in Figure 21-15**



As one can notice, there is a cycle that is created. This cycle is indicative of a potential deadlock situation in the system.

## 21.12 Expanding RDG to include other shared resources

Deadlocks in a network can be caused by other shared elements in the network besides the links and switches. Some of these elements are, but not limited to, buffers used by transactions, and AXI ID reuse at the ATU.

Consider the network example shown in Figure 21-17, where transactions from the Master (M), travel to the I-ATU and via the Fabric to the T-ATU-1. The transaction then continues from T-ATU-1 to M through the customers system. From M it again reaches I-ATU to be directed to go to T-ATU-2.

1. If the AXI-ID issued by M is the same as before:
  - a. I-ATU will assign a sequence number to the transaction.
  - b. A check would be done by the I-ATU if there is enough space in the reorder buffer to buffer the response for this transaction. If there is, the transaction will be

allowed to proceed. If not, then the transaction will be held until there is enough space available. This lack of buffer space can cause a potential deadlock.

- c. If there is enough buffer space, the transaction would be allowed to proceed. However, the response will be held in the I-ATU in the reorder buffer due to ordering conflict with the first transaction. This will cause a deadlock.

In order to identify this potential deadlock and others similar to this one, we include the attributes of the network elements while creating the RDG.

Let us look at the definition of the connection again as described in the previous section.

$C \equiv \{(M_i \rightarrow S_j), (M_i \rightarrow S_k), \dots\} \forall (i \in \text{the connection spans})$  where M is the Master and S is the Slave.

Where each network element in the connection is expanded into its shareable resources. For example, the I-ATU will be in the RDG expanded in to its shareable components, namely: AXI ID, Reorder Buffer Space, Flow Queues, and VC buffers. Similarly, the switch is expanded into its shareable components, namely the VC buffering.

Let us consider the network in Figure 21-17 and let us define the flows as follows:

$$f_1 = \{\text{I-ATU, fabric, T-ATU1}\}$$

$$f_2 = \{T\text{-ATU1, M, I-ATU}\}$$

$$f_3 = \{I\text{-ATU, fabric, T-ATU2}\}$$

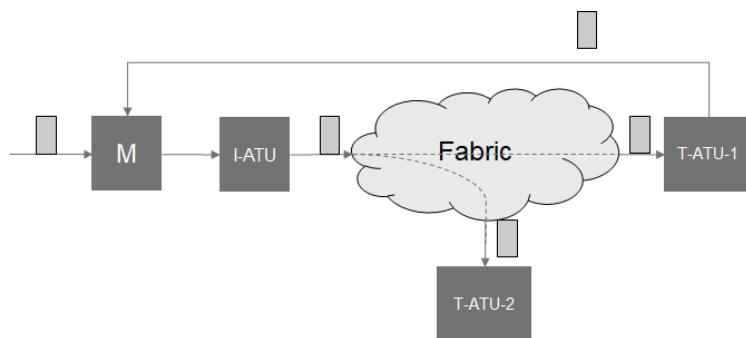
Where the connection and the ATUs are defined as follows :

$$C = \{M, f1, f2, f3\}$$

$$I\text{-ATU} = \{\text{AXI-ID, Reorder Buffer, Flow Buffer, VC Buffer}\}$$

The RDG will determine if there is cycle or not, and then inform the user that there is deadlock or potential to have a deadlock.

**Figure 21-17 System level deadlock due to AXI-ID reuse**



## 21.12.1 Rules

In this section we describe the rules that govern whether a deadlock exists within a cycle or not.

Rule 1:

*If on any Link<sub>i</sub> within the cycle, there exists enough buffer such that the total buffering on Link<sub>i</sub> > Sum of the max length of Packets of the flows mapped to Link<sub>i</sub>, then there is no deadlock.*

Rule 2:

*If along the path of the cycle, there exists enough buffering such that the buffering in the cycle > Sum of the packet lengths of the flows mapped to that cycle.*

## 21.13 Routing Algorithms

Routing algorithm can have a big impact on the performance and overall cost of the Interconnect. There are several routing algorithms that can be used, and based on the type of traffic will result in different performance and network cost.

Routing algorithm development can be segregated in to two broad categories. Ones that fall out naturally from the overall network cost optimization exercise and others that are algorithmic and may have a closed form.

Two algorithms Symphony will use initially are the DoR based X-Y routing and the VAL routing algorithms. X-Y routing results in minimal path distance from source to destination but results in degraded throughput. VAL routing on the hand does not provide minimal path latencies but result in higher throughput in the network.

Two routing algorithms also be combined to create a new routing algorithm. Let R<sub>1</sub> and R<sub>2</sub> be two routing algorithms. We can create a new routing algorithm as follows:

$$R^n(p) = \alpha \times R1(p) + (1 - \alpha) \times R2(p)$$

Where  $\alpha$  could be determined based on type of flow. For example, all latency sensitive flows can be routed using DoR-XY routing and all non-latency sensitive flows routed using the VAL or iVAL algorithm. The path length for the system will also have the same form as the equation above. So will the worst case channel loading.

DOR-XY routing requires 2 virtual channels to make sure no deadlock occurs in k-ary n-cube networks and VAL requires 4 virtual channels, a set of virtual channels for each phase.

# Topology Description and Mapping

## 22.1 Topology Description

Symphony will support regular as well as irregular topologies. In addition to supporting multiple types of topologies Symphony will also support an Interconnect composed of different Fabrics, where Fabrics are connected to each other via specialized Adapters.

The Topology description format has to be versatile enough to support the above requirements.

Symphony has the following primary components:

- ▶ Initiator/Target ATU
- ▶ Switches
- ▶ Adapters (Firewalls, Power adapters, clock adapters, FIFO buffer, Width adapters, etc)
- ▶ Interfaces/Links
- ▶ Power Management Agent (PMA)

A Network Element (NE) is defined as representing any element that has external ports to receive/forward packets/transactions. As such this definition encompasses all of the above listed primary components except Interfaces/links. We define the different NEs as follows:

- ▶ Switch:  $SW_i = \{[\text{Type, Base Class, variations}], \{\text{Ingress Port Map}\}, \{\text{Egress Port Map}\}, \{\text{VC Map}\}, \{\text{Input-Output Map}\}, \{\text{Power Domain, Clock Domain}\}, \{\text{Fabric}\}, \{\text{Input Port-VC Arbitration Map}\}, \{\text{Output Port Arbitration Map}\}, \{\text{QoS-to-VC map}\}, \{\text{Attribute Ptr}\}, \{\text{Location}(x,y)\}\}$  for all “ $i$ ” belonging to Interconnect.
- ▶ Adapter:  $Adp_k = \{[\text{Type, Base Class, variations}], \{\text{Ingress Port Map}\}, \{\text{Egress Port Map}\}, \{\text{Attributes of the Adapter}\}, \{\text{Port Arbitration Map}\}, \{\text{Power Domain, Clock Domain}\}, \{\text{Fabric}\}, \{\text{Location}(x,y)\}\}$  for all “ $k$ ” belonging to Interconnect.
- ▶ ATU:  $ATU_j = \{[\text{Type, Base Class, variations}], \{\text{Initiator/Target}\}, \{\text{Ingress Port Map}\}, \{\text{Egress Port Map}\}, \{\text{Attributes of the ATU}\}, \{\text{Input Arbitration Map}\}, \{\text{Flow-VC Map}\}, \{\text{Power Domain, Clock Domain}\}, \{\text{Fabric}\}, \{\text{Location}(x,y)\}\}$  for all “ $j$ ” belonging to the Interconnect.
- ▶ Interface/Links:  $Links_m/Int_m = \{\text{Reference John's link definition, variations}\}$ .

We define the syntax and semantics of each of the elements that describe them below:

- ▶ {Ingress Port Map}:

- ◆ Is the set of the ingress port connectivity for all input ports of the NE. The elements of the set correspond to the number of inputs of NE, starting with Port 0.
  - ◆ Syntax: [{NE<sub>t</sub>,EgressPort<sub>i</sub>,Int<sub>m</sub>}, .....{NE<sub>l</sub>,EgressPort<sub>m</sub>,Int<sub>t</sub>}]
  - ◆ Where {[String/Num, Num, String/Num]}, NULL value is allowed.
- {Egress Port Map}:
- ◆ Is the set of egress port connectivity for all the output ports of the NE. The elements of the set correspond to the number of outputs of NE, starting with Port 0.
  - ◆ Syntax: [{NE<sub>t</sub>,IngressPort<sub>i</sub>,Int<sub>m</sub>}, .....{NE<sub>l</sub>,IngressPort<sub>m</sub>,Int<sub>t</sub>}]
  - ◆ Where {[String/Num, Num, String/Num]}, NULL value is allowed.
- {VC Map}:
- ◆ VC Map describes how the VCs at the input ports are connected to the output ports. Normally all the input VCs will be connected to all the output ports, but there may be instances where some VCs on the input may not be connected to the some output ports. VC map is an Nx(KxM) table that gives the capability to enable exclusion, where there are N inputs, each input has K VCs and there are M output ports. Each VC Map enumerates the exclusions for that VC only.
  - ◆ Syntax:  
[{{<comma separated exclusion list for input 0, VC0>}}, {<...>}, {{<comma separated exclusion list of input 0, VCn>}}]  
[{{<comma separated exclusion list for input 1, VC0>}}, {<...>}, {{<comma separated exclusion list of input 1, VCn>}}]  
.....  
[{{<comma separated exclusion list for input N, VC0>}}, {<...>}, {{<comma separated exclusion list of input N, VCn>}}]
  - ◆ Where NULL value is allowed. If there are no exclusions then a single NULL entry within the [Null] is sufficient to indicate it.
- {Input-Output Map}:
- ◆ NxM Exclusion table of which inputs are not connected to what outputs, where there are N inputs and M outputs.
- {Power Domain, Clock Domain}:
- ◆ Power and clock domain that the NE belongs to. Exceptions are the Power Domain/Clock domain adapters which get their power/clock information from the links they are attached to.
  - ◆ Where each element is {String/Num}.
- {Fabric}:
- ◆ The Fabric the NE belongs to. Note: The interconnect may consist of multiple Fabrics.
- {Input Port-VC Arbitration Map}:

- ◆ This is a map of which arbitration algorithm is used for arbitrating within the VCs on an input port. Input Port-VC arbitration map is only needed for a Buffered VC switch. The Map is a Nx(M+1) table which enumerates the type of arbitration algorithm and its parameter values. N is the number of inputs and M is number of VCs. This map is needed only when VCs are implemented.

◆ Syntax: [{Type of arbitration policy}, {weight VC<sub>0</sub>, weight VC<sub>1</sub>, ....weight VC<sub>K</sub>}].

✓ Note: For Round Robin arbitration scheme the weights would be all 1. For priority arbitration weights are not needed at all.

► {Output Port Arbitration Map}:

- ◆ This is a map of which algorithm/s is/are used for arbitrating the output port. When VCs are implemented the arbitration is performed by a 2-stage hierarchical scheduler. When VCs are not implemented the arbiter is a single stage arbiter.

◆ Syntax: [{Output Port, Type of Arbitration policy for VC-arbiter, (weight Input<sub>0</sub>, weight Input<sub>1</sub>, ....weight Input<sub>K</sub>), Type of Arbitration policy for the Master Arbiter, (weight VC<sub>0</sub>, weight VC<sub>1</sub>, ....weight VC<sub>K</sub>)}].

► {QoS/Tier-to-VC Map}:

◆

► {Type, Base Class, variations}:

- ◆ The type field defines the type of the network element, whether it is a switch, adapter (which kind of adapter), which type of ATU.
- ◆ Base Class defines the kind of NE. Where variation define restrictions and Increments from the base class.
- ◆ Syntax: {String <Power domain, Clock Domain, Width Adapter, FIFO, ....>, reference(pointer, reference(pointer)}

► {Port Arbitration Map}:

- ◆ The adapter is a single input-output NE. As such the port arbitration is a single level arbiter that arbitrates on the VCs.
- ◆ Syntax: [{Type of Arbitration policy for the Master arbiter}, {weight VC<sub>0</sub>, weight VC<sub>1</sub>, ....weight VC<sub>K</sub>}]

✓ Note: For Round Robin arbitration scheme the weights would be all 1. For priority arbitration weights are not needed at all.

► {Input Port Arbitration}:

- ◆ An ATU could be defined as a multi-ported ATU. This field defines the type of arbitration used and the associated parameters thereof.
- ◆ Syntax: [{Type of Arbitration policy for the Master arbiter}, {weight VC<sub>0</sub>, weight VC<sub>1</sub>, ....weight VC<sub>K</sub>}]

► {Flow-VC Map}:

- ◆ This defines how the Flow queues in the CT Layer of the ATU map to the VCs in the

► {Attribute Ptr}:

- ◆ Attribute ptr points to file where the attribute data for the NE is stored. Attributes are defined later in the document.
- ▶ {Location (x,y)}:
  - ◆ This field defines the physical location where the NE is placed on the SoC.

## 22.1.1 Customer IP, Port and SoC Residual Space Capture

NoC placement requires that the Residual Space (RS) on the SoC be captured along with the locations of the ports that connect the customer IP to Symphony NEs. Residual space is space on the SoC where the customer would like to place the NoC.

The Residual Space is represented by a graph  $G_{RS}(V,E)$  where “V” is a set of vertices and E is a set of edges that connect these vertices. The resulting graph forms a polygon, i.e., it is a plane bounded by finite chain of straight lines closing in a loop to form a circuit. The vertices are represented by location in the (x, y) coordinates.

It is also important that the port which connects the Customer IP to Symphony NE be identified along with their location. These ports have to lie somewhere on  $G_{RS}(V,E)$ . The ports are defined as follows:

- ▶ Port<sub>i</sub>:= [{Location(x,y)}, {CustomIP Name}, {Symphony NE}, {Link}]
  - ◆ where, location is the location of the port.
  - ◆ CustomIP Name is the name of the customer IP that the port is connected to.
  - ◆ Symphony NE is the NE that the port is connected to on the Symphony side.
  - ◆ and Link is the attributes of the link that connects the port to the Symphony NE.

## 22.1.2 Defining Hierarchies

Hierarchies are defined recursively as follows:

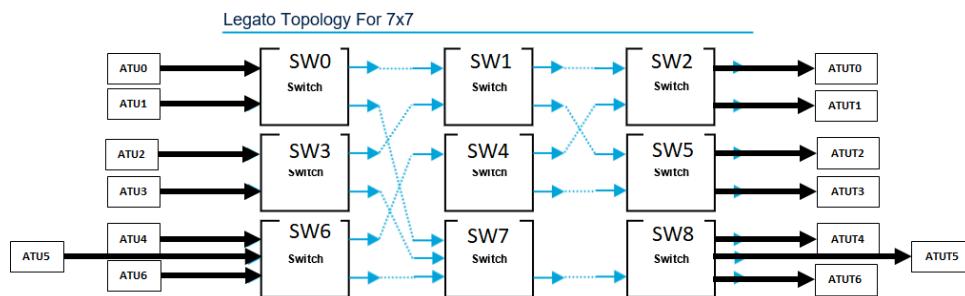
Parent\_Module<sub>j</sub>: {Child<sub>0</sub>, Child<sub>1</sub>,.....Child<sub>n</sub>}

Child<sub>j</sub>:{...,,...,..}

The recursion ends at the NE level, as defined in the previous sections. Port definitions and connectivity information is carried at the NE level.

## 22.1.3 Example

Figure 1 below shows the topology that Legato will use for the Data network to support Integration Event 1.



**Figure 22-1 Data network for NCORE IE1.**

Below is how the different elements would be described by the topology format discussed above.

ATU0 = [{Packetizer; Pkt\_0; NULL}, {Initiator}, {AIU0, 0, SMI\_Data}, {SW0, Port0, Link Ptr}, {ATU0 attribute Ptr}, {NULL (no VCs)}, {PDO. CD0}, {Legato\_Fab\_Data\_7x7}, {Location(x,y)}]

SW1 = [{2x2 Switch, 2x2 S\_base\_class, NULL}, {(SW0, Port0, Link Ptr), (SW3, Port0, Link Ptr)}, {(SW2, Port0, Link Ptr), (SW5, Port1, Link Ptr)}, {NULL (no VCs)}, {Null Matrix (fully connected switch)}, {PDO, CD0}, {Legato\_Fab\_Data\_7x7}, {NULL (no VCs)}, {(Port 0, PRI/WRR/RR, (w<sub>0</sub>=x, w<sub>1</sub>=y)), ((Port 1, PRI/WRR/RR, (w<sub>0</sub>=t, w<sub>1</sub>=z))}, {NULL (no VCs)}, {SW1\_Attribute\_Ptr}, {Location (x,y)}]}

The Target ATU can be defined in a manner similar to that of the Initiator ATU.

## 22.2 Network Element Attributes

The description of the different network elements that Symphony has is given in the earlier sections. This section outlines the attributes of these different network elements.

NE attributes will include the following:

- ▶ Latency
- ▶ Pipe stages
- ▶ Timing and other characterization data,
- ▶ Logic size
- ▶ Power

## 22.3 Rules and Restrictions

This section defines the rules and restrictions the different Symphony and NCORE Network Elements. Release restrictions will be indicated where necessary.

## 22.3.1 Packet Format:

**Table 22-1 Packet Header Restrictions**

Item No.	Restriction/Rule	Release No.	Maestro Visible	Comments
1	Symphony Header format will be fixed with the Fabric. Different Fabrics within the Interconnect can have different Header formats.	Legato 1.0 Presto 1.0	Yes	Header formats for all packet types: Request/Response; Control; Error would be the same per Fabric. If control/response/error packets travel over different fabrics than the header formats may be optimized.
2	NDP Message should always be the last field of the packet header.	Legato 1.0	No	Currently this restriction only applies to Legato 1.0. Presto does not have NDP. Post Legato 1.0, NDP may be moved out of the header field and carried separately.
3	Header field positions will not change from link to link in a given fabric.	Legato 1.0 Presto 1.0	No	This condition is true for all releases.
4	No Header field will be split, except for the address field.	Legato 1.0 Presto 1.0	This condition impacts minimum link width	
5	Header shall always be the first PHIT for primary fabrics.	Legato 1.0 Presto 1.0?	No	
6	Base packet format is described here: <add link to “Bob the Builder”>	Legato 1.0 Presto 1.0	Yes	

## 22.3.2 Packetizer and Depacketizer

**Table 22-2 Packetizer/depacketizer rules**

Item No.	Restriction/Rule	Release No.	Maestro Visible	Comments
1	Messages on the SMI interface from the I-AIU are not interleaved. That is AIU will finish one message before starting to transmit another.	Legato 1.0	No	This restriction does not hold for Symphony in general. However, for Presto, POR is that the I-ATU will only support a single VC. The T-ATU will support multiple VCs.
2	No VC support in the Packetizer/Depacketizer	Legato 1.0	Yes	
3	Single VC support at the I-ATU and multiple VC support at the T-ATU	Legato 1.1 Presto 1.0	Yes	Future release of Symphony will have multi-VC support at I-ATU also. VC support (Phit level multiplexing) will be available in Presto 1.0 time frame.

4	A single NCORE element can have multiple SMI ports. Each SMI port will attach to a single Packetizer/Depacketizer.	Legato 1.0	Yes	NCORE will define the number of physical Fabrics that are needed. Each Fabric will have separate SMI interface and a dedicated Packetizer and Depacketizer module attached to it. Maestro will have to instantiate a packetizer/depacketizer for each Fabric.
5	SMI interface with or with Data Payload	Legato 1.0	Yes	
6	No protection scheme employed by the transport. All protection will be end-to-end and done at the AIU, except in power/clock adapters and Firewalls.	Legato 1.0	Yes	Transport will not add any protection such as ECC or parity, except on the Route field. Legato may put parity protection on the route bits.
7	Header format: - Header Parallel (HP=0) - zero latency configuration. - Header "full" serial (HP=1) - 1 cycle latency	Legato 1.0	Yes	In Header Parallel, the interface width is = Sum(Header + Data), that is, the interface width will match SMI interface width. In Header "full" Serial, there would be 1 cycle latency and the interface width is = Max(Header, Data).
8	Header format for configuration network only: - Header Serial	Legato 1.0	Yes	Header serial format means that the header can be contained in more than one beat on the link.
8	Route bits are calculated based on the Target ID and the Steer field	Legato 1.0	Yes	Route calculation: $R(\text{TargetID}, \text{Steer}) \rightarrow \{\text{route bits}\}$
9	Protection Scheme used and the fields covered: Protection Scheme Used: - Parity - ECC Fields Covered:	Presto 1.0 Legato 1.0	Yes	For Legato 1.0, Symphony protection will be limited to route bits. NCORE 3.0 will have end-to-end protection which other than in error conditions would be oblivious to Legato. For Presto, Symphony will protect all the important fields that the systems want to be protected.
10	A single AIU/ATU can have multiple ports connected to the same Fabric via multiple Packetizers/Depacketizers	Legato 1.0 Presto	Yes	

### 22.3.3 Link Format

**Table 22-3 Link Format Rules**

Item No.	Restriction/Rule	Release No.	Maestro Visible	Comments
1	Link format can change from one link to another, however, the relative field positions will not change within a Fabric	Legato 1.0 Presto 1.0	Yes	
2	Link format and width is the same at both ends of the link. If the width and/or format is different, then an adapter has to be inserted in the middle, splitting the link in two separate links.	Legato 1.0 Presto 1.0	Yes	

3	Both ends of the link should be in the same clock and power domain	Legato 1.0 Presto 1.0	Yes	
3	Link format details: <add link to “Bob the Builder”>			

## 22.3.4 Switch

**Table 22-4 Switch Rules**

Item No.	Restriction/Rule	Release No.	Maestro Visible	Comments
1	Non-VC Aware Flow Through Switch	Legato 1.0 Presto 1.0	Yes	
2	VC Aware Flow-Through Switch	Presto 1.0	Yes	
3	Arbitration Schemes available: - Priority - Weighted Round Robin - Round Robin	Legato 1.0 Presto 1.0	Yes	
4	All ports (ingress/egress) of the switch have the same width. Any width change on the link would need a width adapter.	Legato 1.0 Presto 1.0	Yes	

## 22.3.5 Power and Clock Adapter

**Table 22-5 Power/Clock Adapter rules**

Item No.	Restriction/Rule	Release No.	Maestro Visible	Comments
1	A power/clock domain adapter has to be inserted to connect two different power and clock domains within the Fabric/Interconnect.	All	Yes	
2	Clock/power domain adapter is a three ported adapter. The third port connects to the return network. The clock/power domain adapter is not bidirectional and does not need to store state regarding messages flowing in the forward direction.	Legato	Yes	

3	Clock/power domain adapter for Presto 1.0 will be bidirectional  - If Clock/power domain adapter is bidirectional, then the response path for all requests flowing through the adapter also has to flow through the adapter. The adapter holds the state of each request/response that flows through the adapter. It is also assumed that the this adapter assists in determining when not to clock/power down the domain.	Presto	Yes	Future versions of Presto may enable requests which have forwarding capabilities. That is, the response for the request may be sent to some other NE, other than the initiator of the request. This scheme will be useful for DMAs and help reduce latency.
4	For NCORE, the adapters will return a fully formated the error packet.	Legato 1.0	No	

## 22.3.6 Width Adapter:

**Table 22-6 Width Adapter Rules**

1	Insert Width Adapter when two Network Elements connected together by a point to point link don't have matching widths	Presto Legato	Yes Yes	
2	ATP interfaces on both sides should have he same base Packet definition	Presto Legato	Yes Yes	
3	Add width adapter when the packet styles on the two connecting interfaces are of different styles	Presto Legato	Yes Yes	For example: When going from 32 bit ATP link with Serial packet style to a 32 bit ATP interface with parallel packet style, insert a width adapter



## 23.1 VC Aware Flow Through Switches

Symphony Flow Through switches are bufferless switches in which PHITs propagate the network as far as they can by winning output port arbitrations along the way, until they reach a Network Element which can buffer the PHIT and issue a Ready signal. The ready signal then propagates back to the source.

The PHIT is considered transferred when both the Source and Destination see the Valid and Ready signal asserted.

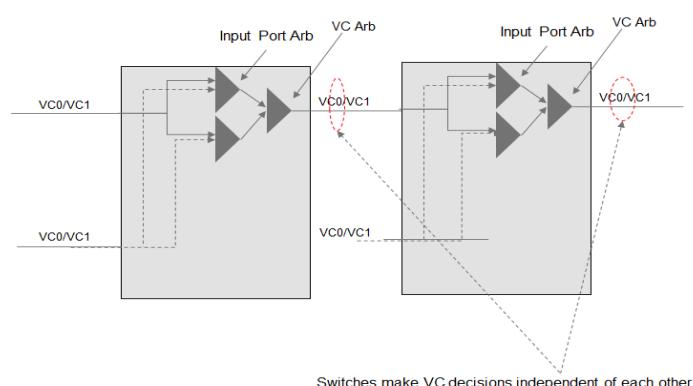
As the PHIT makes progress through the network, switches along the way lock the output port arbiter. The lock remains until the switch sees the last PHIT for the packet. FT switches use wormhole routing.

The output arbitration decision at each switch is made without taking into account the state of the downstream switch.

The FT-switches are simple and offer the best PPA.

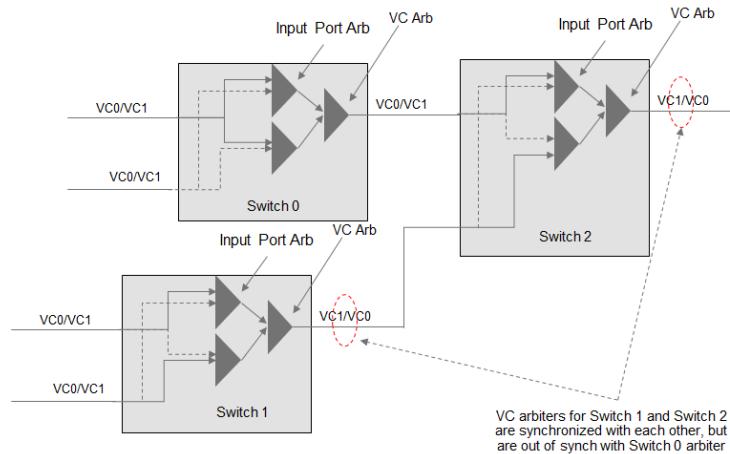
This Flow Through concept is extended to Virtual Channels, by using a vector of Valid/Ready signals to represent the Virtual Channels and their respective Ready signals. At any given point in time, only one Virtual Channel can be active on the link. Once a packet occupies a VC, no other packet can be on the same VC until the earlier packet is done transmitting, that is the switch will lock the packet to the VC until it sees the last PHIT.

In VC aware FT switches, PHITs from different VC can interleave on the same channel. The switch output port arbiter is a two stage arbiter. The first stage arbitrates between the inputs that are competing for the VC on that output port. The second stage then arbitrates between the valid VCs to choose a winner. A VC is valid for arbitration only if it has PHIT to transmit. The VC arbitration decision is taken independent of the state of the downstream switch. Figure 23-1, illustrates this scenario.



**Figure 23-1 : VC arbitration between two switches**

Since the VC arbitration decision in the current switch is taken independent of the state of the downstream switch, there could arise situations where in a fully loaded always backlogged network, the arbitration scheme in the current and the immediate downstream switch chose different VCs to service in such a manner that they are always out of sync. We illustrate this scenario in Figure 23-2, below:



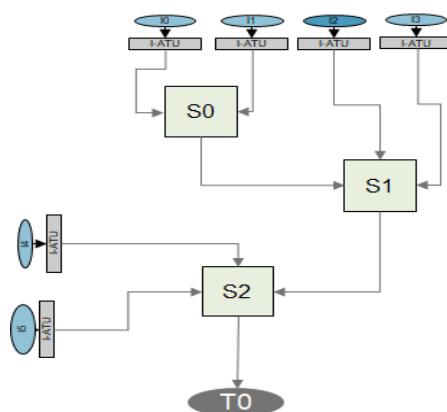
**Figure 23-2 : In a fully loaded network, the above scenario can happen. The VC arbiter belonging to Switch 1 and Switch 2 are in synch but out of synch with the VC arbiter of Switch 0.**

The above situation will prevent packets from Switch 0 from making forward progress in a predictable manner. We refer to this effect as “VC-Shutout”.

We ran multiple simulation in order to investigate this issue. We discuss these results in the next section.

### 23.1.1 Simulation Results

The simulation set up is shown in Figure 23-3 below. It consists of 5 Initiators (Source), 3 switches and a single Target (Destination). The network supports 2 Virtual Channels. Each Initiator only sends traffic on one VC only. The start times of Initiator 1 & 2, are delayed by a few cycles. Simulation experiments were conducted using the Round Robin (RR) and Weighted Round Robin (WRR) algorithms to arbitrate between the Virtual Channels. The target sink rate was varied from 100% that of the link rate to 50% of the link rate.



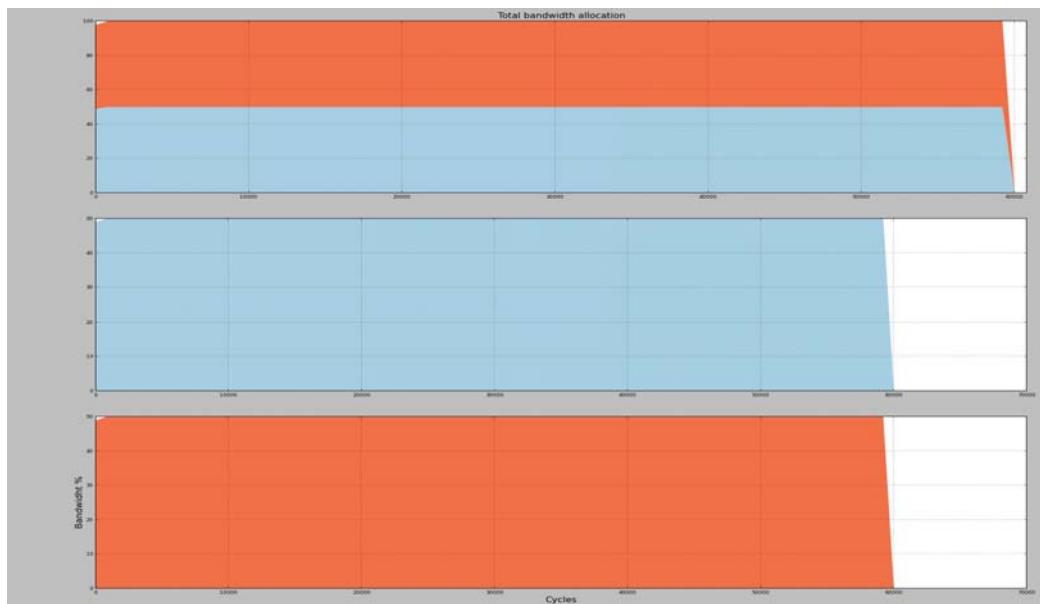
**Figure 23-3 : Simulation setup**

Case 1:

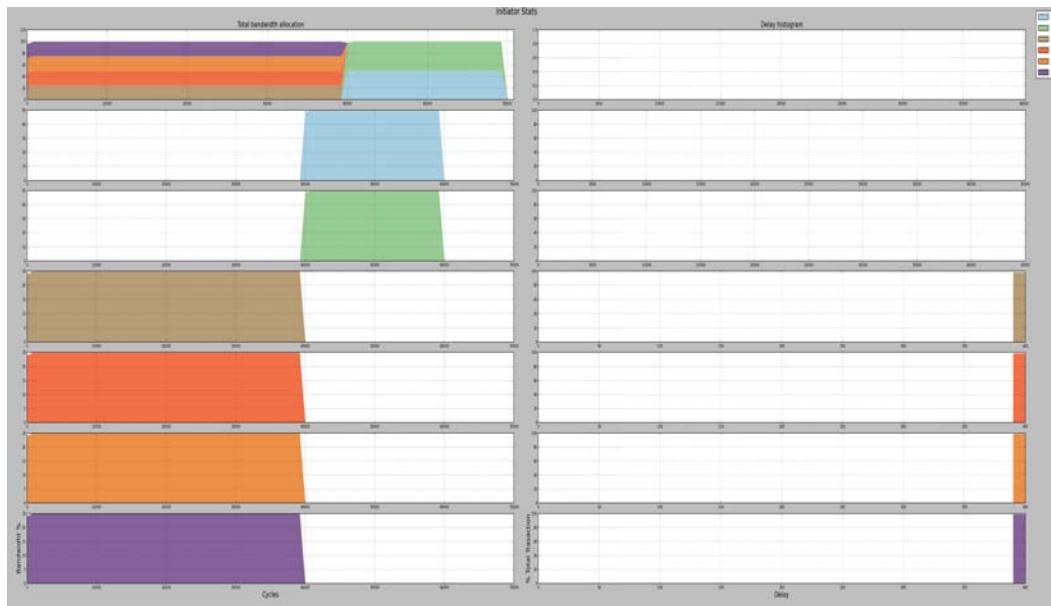
**Table 23-1 Setup values for Case 1**

Item	Value/Comments
Offered Traffic Per Initiator	100% of the link rate
Target sink rate	100% of the link rate
Arbitration Algorithm	Round Robin
Late start	Initiator 0 and Initiator 1
VC allocation	Init 0, 2, & 4 transmit on VC0 Init 1, 3, & 5 transmit on VC1

The Round Robin (RR) algorithm will move on to a different VC every cycle in a round robin fashion as long as that VC has a valid PHIT to transfer. The RR policy should equally distribute bandwidth between the VC 0 and VC1. Figure 23-4, below shows how the bandwidth of the link connecting all the switches is distributed between the VCs.

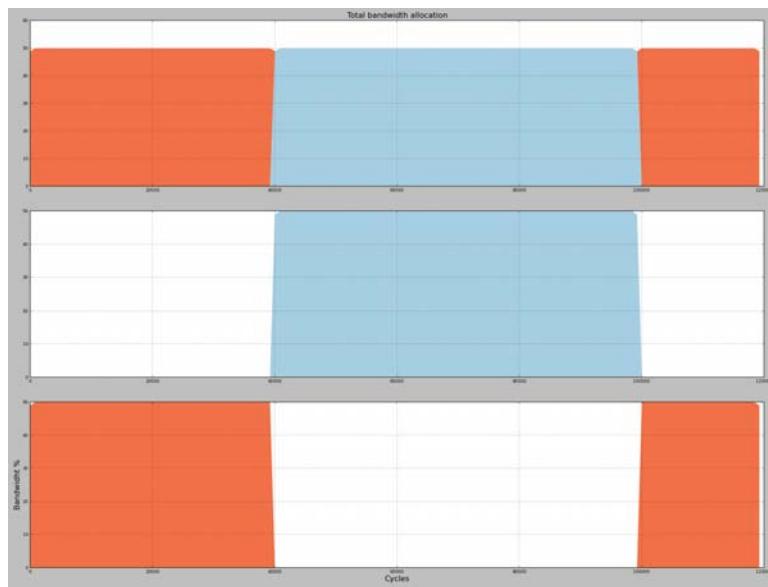
**Figure 23-4 Bandwidth Allocation per VC: Red VC1 and Blue VC0**

The system distributes bandwidth equally between the VCs as expected, however, looking at how bandwidth is distributed between the Initiators (Figure 23-5), we can see the VC-Shutout effect impacting Initiators 0 & 1. Packets from Initiators 0 & 1 are not able to make forward progress until the rest of the initiators have finished transmitting.

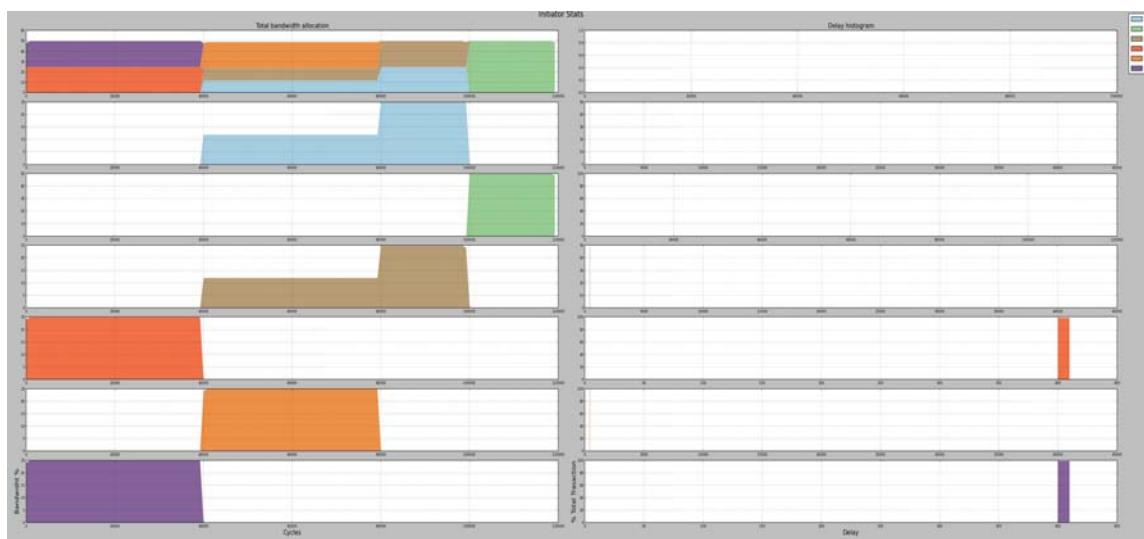


**Figure 23-5 : Bandwidth allocation per Initiator. Initiators 0 & 1 cannot make forward progress until the rest of the initiators are done transmitting.**

The “VC-Shutout” effect is even more prominent when the target drain rate is reduced to 50% of link rate as illustrated in Figure 23-6 & Figure 23-7.



**Figure 23-6 : VC bandwidth allocation when the Target drain rate is 50% of the link rate.**



**Figure 23-7 Initiator bandwidth allocation when Target drain rate is 50% of the link rate. Initiators 0, 1, 2, and 4 cannot make forward progress until initiators 3 & 5 are done.**

As is evident from Figures above certain initiators are shut out and cannot make forward progress because of the VC-Shutout effect.

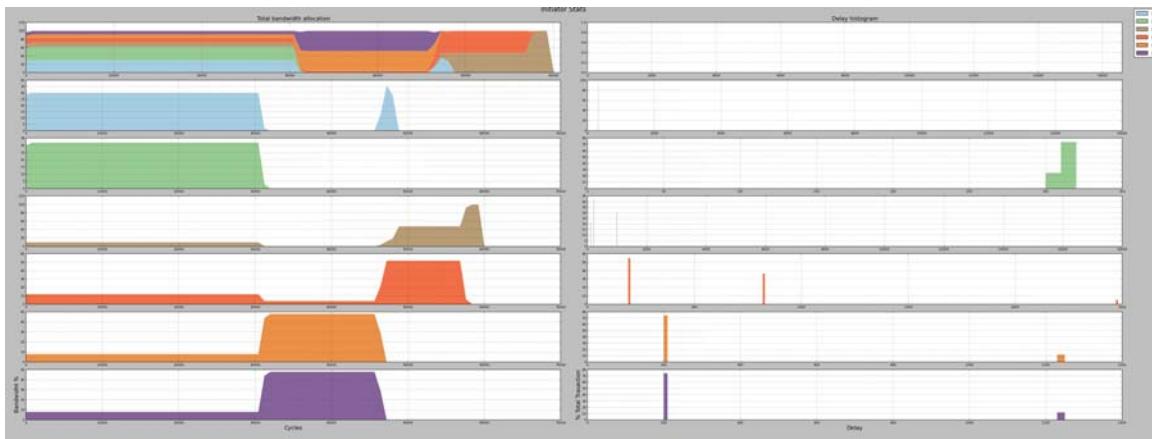
Next we look at the Weighted Round Robin (WRR) algorithm.

Case 2:

**Table 23-2 Setup values for Case 2**

Item	Value/Comments
Offered Traffic Per Initiator	100% of the link rate
Target sink rate	100% of the link rate
Arbitration Algorithm	Weighted Round Robin
Late start	Initiator 0 and Initiator 1
VC allocation	Init 0, 2, & 4 transmit on VC0 Init 1, 3, & 5 transmit on VC1

In Figure 23-8 below, the results for the above setup are presented.



**Figure 23-8 : WRR arbitration scheme. Weights 1:1 to equally divide bandwidth between initiators at the same level.**

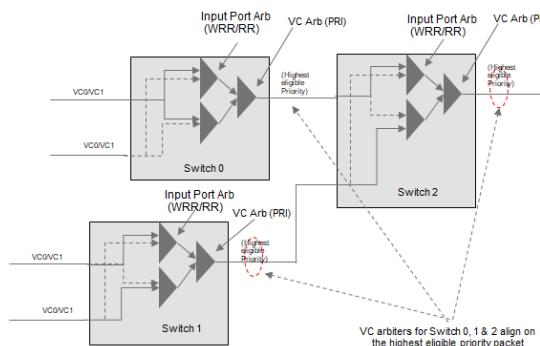
Although the bandwidth is equally divided between the initiators at the same level, it is unstable and does not have predictable forward progress.

Several other experiments were done with varying traffic loading. Some yielded better results than presented here. However, since we cannot control the exact conditions the Symphony solution would be deployed in, we cannot rely on these restrictions.

## 23.1.2Solutions:

### 23.1.2.1Priority based VC-aware Flow Through Switches

The VC-Shutout does not impact priority arbitration. In priority arbitration, the arbiters naturally align based on the priorities of the eligible VCs, hence the “VC-Shutout” does not happen. Bandwidth between flows/traffic coming from different inputs and converging on a single VC can be apportioned using the first stage WRR/RR as shown in Figure 23-9 below.



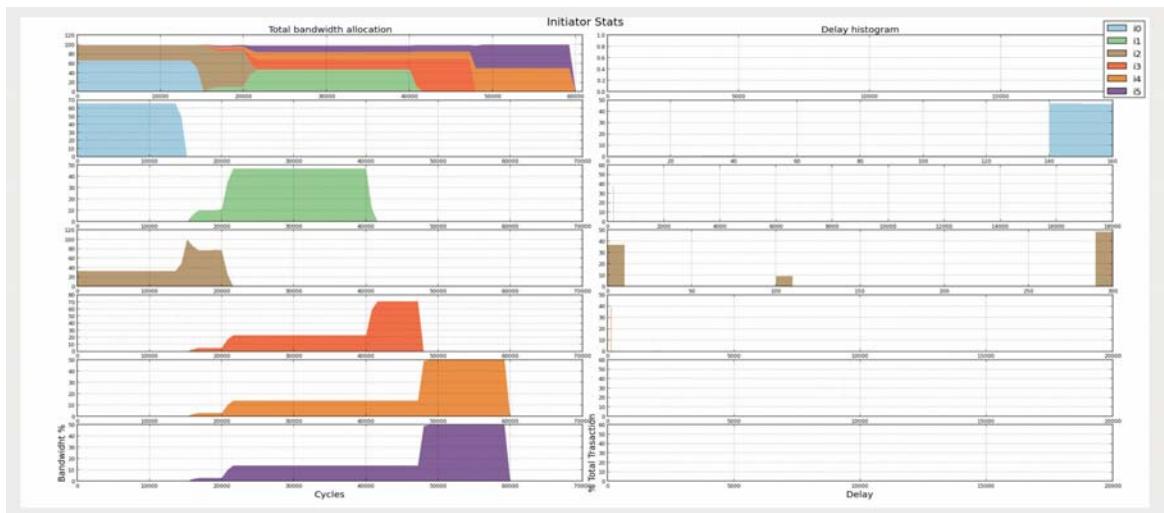
**Figure 23-9 Priority VC Arbitration network**

Simulations were run to determine if the “VC Shut-Out” happens with priority VC arbitration. The simulation network was the same as shown in Figure 3. The rest of the setup was as follows:

**Table 23-3 Simulation setup for Pri-VC arbitration case**

Item	Value/Comments
Offered Traffic Per Initiator	100% of the link rate
Target sink rate	100% of the link rate
VC Arbitration Algorithm	Priority
Input port arbitration (first level arbiter)	WRR
Late start	Initiator 0 and Initiator 1
VC Priority allocation	Pri 3 (HI): VC0; Pri 0 (Low): VC1
VC allocation	Init 0 & 2 transmit on VC0 Init 1, 3, 4 & 5 transmit on VC1
Input Port Arbiter (on the egress port)	WRR: Init 0: Init 2:: 2:1 Init 1: Init 3::2:1 Init 4: Init 5::1:1

Figure 23-10, below shows the bandwidth allocation for different Initiators on the link. Notice that the high priority initiators get the bandwidth that they need. The bandwidth of the high priority initiators is divided in the correct ratio of 2:1. Once the high priority initiators are done transmitting, the low priority initiators start transmitting. The bandwidth between the low priority initiators is also divided according to the weights assigned.

**Figure 23-10 Pri VC arbitration and WRR allocation for the input ports**

In Figure 23-11, the drain rate of the target was reduced to 50% of the link rate. All conditions remain the same except that there are 4 VCs in the system. Flows from initiator 0 & 2 are still mapped to Priority 3 and VC0. Flows from initiator 3 are mapped to Priority 2 and VC 1. Flows from initiator 1 & 4 are mapped to Priority 1 and VC2 and finally flows from Initiator 5 are mapped to Priority 0 and VC 3. Comparing Figure 17, with Figure x earlier, it is clear that the behavior of the network with priority is predictable whereas the behavior of the network in Figure x with WRR arbitration was not predictable.

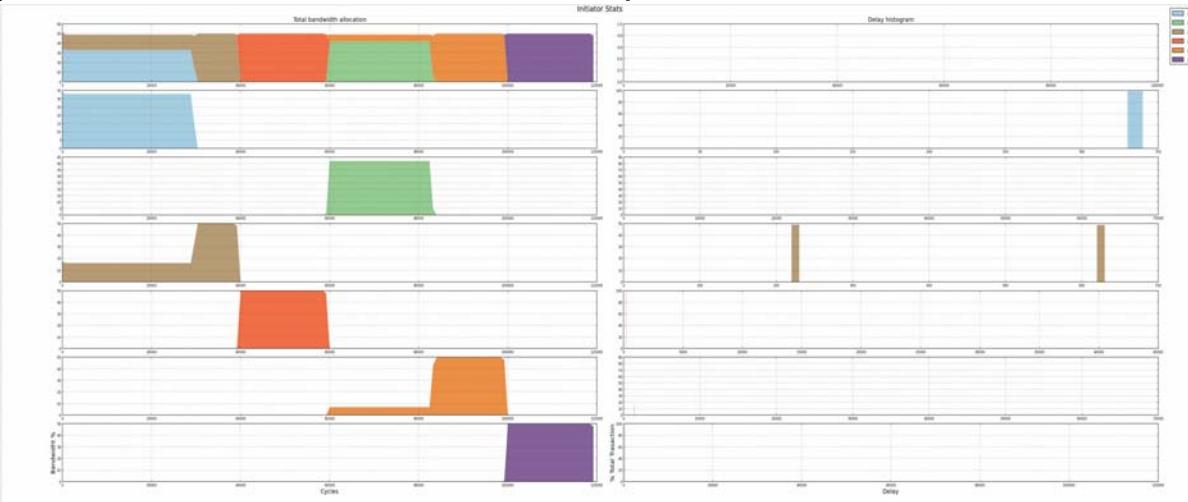
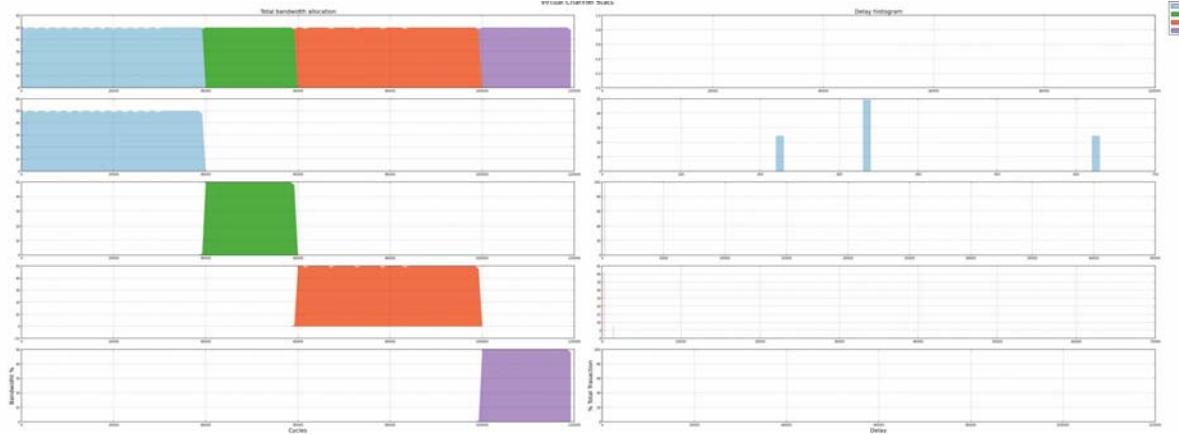
**Figure 23-11 Pri VC arbitration and WRR allocation for the inputs when the Drain rate is 50%**

Figure 23-12, below presents how the bandwidth is allocated to the VCs.

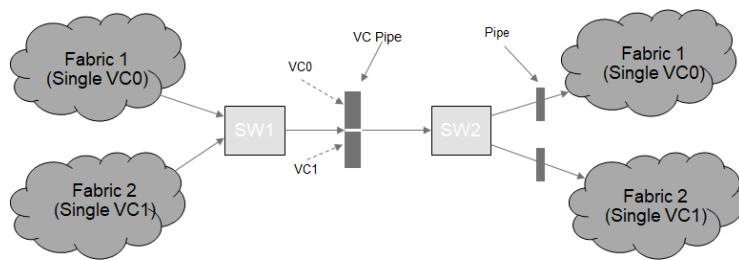
**Figure 23-12 Bandwidth Allocation per VC**

Several other simulation tests were done in order to make sure that the VC-Shutout phenomenon does not happen or how the priority flows effect overall network performance. Priority flows can cause starvation for lower priority flows and hence need to be regulated by credit and I-ATU rate limiting mechanism.

### 23.1.2.2 VC-aware Flow Through Switches Restricted Usage

In the section above, we have shown that Priority VC arbitration works with VC aware flow through switches. In this section we would analyze whether there are scenarios, restricted as they may be, where WRR and RR algorithms can be used with Flow Through VC-Aware switches.

One such case is shown in Figure 23-13 below:

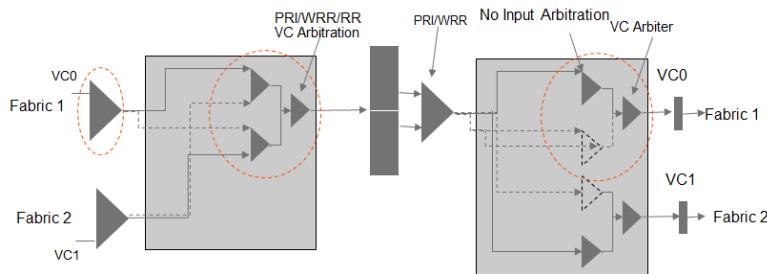


**Figure 23-13 Network configuration: Usage of VC-Aware Flow Through Switch**

In Figure 23-13, the VC-Lite adapter (Pipeline) will allow different Fabrics to share a single physical set of wires. The following are the conditions under which such an arrangement would work:

- ▶ Single VC Fabrics: Traffic from each Fabric will be mapped to a different VC. Either priority or WRR VC arbitration can be used in the Switches and Adapters connecting the two Fabrics.
- ▶ Priority VCs in the Fabrics: Traffic from each Fabric will be mapped to the corresponding VC in the switches and adapters. The bandwidth of the link is partitioned between the two fabrics using WRR algorithm on each of the input arbiter. VC arbitration has to be priority based in this case.

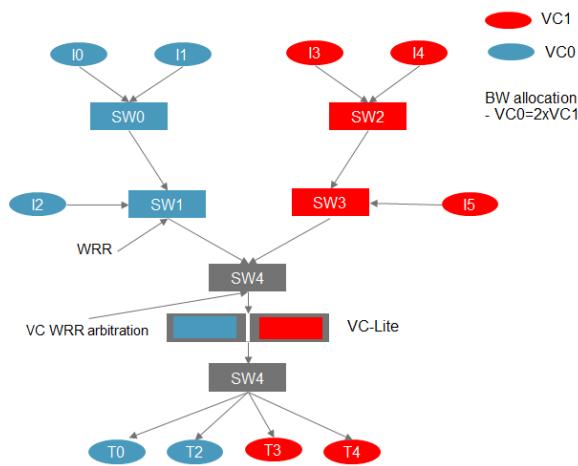
Figure 23-14, illustrates the switch configuration for network shown in Figure 23-13 above.



**Figure 23-14 : Switch and VC Lite configuration for the network in Figure 23-13.**

The reason why such an arrangement works is because all traffic enters the VC-Lite adapter and the Switch 2 through a single interface. There is no competing traffic at these network elements. The VC arbiter at SW2 is forced to accept whatever is presented to it by the VC-lite adapter.

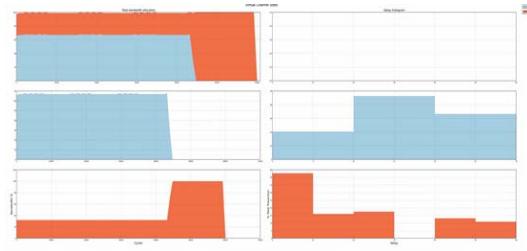
Network simulations were run to validate the above conclusions. The network that was simulated is shown in Figure 23-15.



**Figure 23-15 Simulation network**

The blue network transmits on VC0 only and the red network transmits on VC1. Since there is only a single priority in the network and single VC, WRR VC arbitration algorithm is used at SW4 to partition the link bandwidth. The bandwidth at SW4 is partitioned between the blue and the red network in a ratio of 2:1.

Simulation results are presented in Figure 23-16 below. The blue graph is for VC0 and the red is for VC1. The bandwidth allocation is clearly in 2:1 ratio.



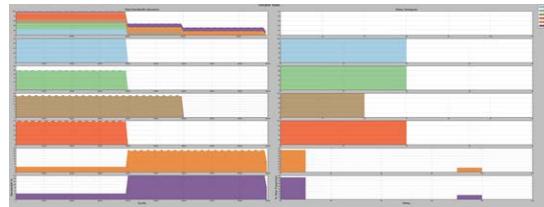
**Figure 23-16 Bandwidth distribution between the two VCs based on WRR arbitration**

Another simulation was run to evaluate the impact of priority flows on the network. The simulation setup is described in the Table below.

**Table 23-4 Simulation setup**

No of VCs in each Fabric	2
Priority Allocation	High::VC0, Low::VC1; Init0, Init1, Init3 – VC0; Init2, Init4, Init5 – VC1
Bandwidth Allocation	Blue net vs. Red net::2:1
Latency	20 cycles
Credit Control	High Priority flows vs. Low priority flow::3:2

Simulation results are shown in Figure 23-17. From the figure it is clear that high priority initiators are given priority over low priority initiators. The bandwidth between the networks is also partitioned in the 2:1 ratio.

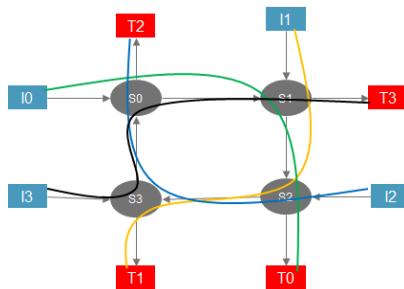


**Figure 23-17 Bandwidth distribution between different Initiators with priority flows**

### 23.1.2.3 Full blown VC Switch

### 23.1.3 Case of the Ring Network

Deadlock and deadlock avoidance would be discussed elsewhere. However, here it suffices to state that a deadlock would be created whenever there is a cycle in the Resource dependency graph. In order to prevent deadlocks from happening, VCs are used to create an escape path. Consider the network in Figure 23-18 below.



**Figure 23-18 Example of a network that will create cycles in the RDG**

If a RDG was drawn for the above network and the flows, it will show a cycle being created. This cycle can lead to a deadlock. In order to remove the deadlock an escape path needs to be added to one of the flows so that the dependency cycle breaks. One way would be to add a VC to the link between switch S0 and S1 and route traffic from I3-T3 to flow over that VC on that link. For the VCs to work on the S0-S1 link and partition the bandwidth in a fair manner, we need to implement the WRR VC arbitration algorithm. Hence for situations like one described here, we need a full blown VC switch to support regular as well irregular networks. The reason why adding a VC resolves the issue is because of PHIT interleaving. Packet interleaving will not remove the deadlock.

## 23.2 Conclusion

Based on the above analysis, our conclusions are as follows:

- ▶ VC Shutout is a problem with the FT-VC aware implementation only if WRR/RR arbitration for choosing the VC is used.
- ▶ WRR/RR VC arbitration would work in the VC-Lite and VC-Links use case.

- ▶ Need WRR and RR arbitration for providing a general solution as well as arbitration when VCs are used to remove cycles within the RDG.
- ▶ A Pipelined/buffered VC switch will solve issues highlighted in the analysis.

# 24

## Multicast

### 24.1 Multicast

Multicast support in NoCs is gaining importance given the use of multicast in Coherency protocols and new and emerging applications such as machine learning.

Although one-to-one (unicast) traffic will still be the dominant traffic for most applications, the impact of multicast traffic, if not handled properly, on unicast traffic performance is many times the actual multicast traffic load. In [VMT], it was shown that for a 4x4 mesh packet switched network with back pressure, as the percentage of multicast traffic (transmitted as 'n' unicast) is increased from 0% to 10% the carried network load reduces from 50% down to 15%. This is because the multicast messages were sent as unicast messages to the chosen multicast destinations. Note that the multicast destinations and the size of the multicast group was chosen randomly. Increasing multicast traffic beyond 10% further reduced the overall carried load in the network.

In a system where a request from a Master is followed by a response from the Slave, a single multicast message sent by the Master to 'n' slaves generates 'n' responses. These 'n' responses if not handled properly can cause congestion on the return path to the Master. Hence multicast traffic in request-response networks effects the network's ability to carry traffic in both directions.

The symphony architecture solution described in this section addresses both types of multicast traffic, which we will refer to as the FanOut and FanIn traffic from hereon.

This section will deal with the following:

1. Goal of multicast support in Symphony
2. Multicast routing and deadlock avoidance
3. I/T-ATU support for multicast FanOut
4. I/T-ATU support for multicast FanIn
5. Switch support for multicast FanOut
6. Switch support for multicast FanIn
7. Flow Control
8. Error conditions
9. Future designs (multicast lookahead)

- Areas we need to look into:

- ▶ Routing for regular grid type networks, multi-way tree based networks, Rings, and non regular networks
  - a. table based
  - b. non-table based
- ▶ Deadlock detection and removal for multicast traffic
- ▶ I-ATU/T-ATU implications

- ▶ Switch implications
- ▶ Flow Control
- ▶ ATP protocol implications and additional fields.
- ▶ Performance evaluation: Impact of multicast traffic on the performance of the Fabric.

## 24.2 Multicast Routing

Symphony shall support the path based as well as tree based multicast routing. In the path based multicast routing, the packet is forwarded to the first node in the multicast list. The node then forwards the packet to the next node in the multicast list and so on and so forth. In Tree based multicast routing all the paths from the source to 'n' multicast destinations are assembled in the form of a tree.

The Path based multicast is more latency prone as the packet serially traverses the destination in the multicast group. However, for supporting multicast in the ring, Symphony will use modified path routing as will be explained later.

### 24.2.1 Tree based Forward Multicast Routing

Symphony shall support three flavors of tree based forward multicast routing algorithms. Forward multicast represents routing from Source (initiator) to the Destination (Target) nodes. Reverse multicast routing represents routing for the merge network.

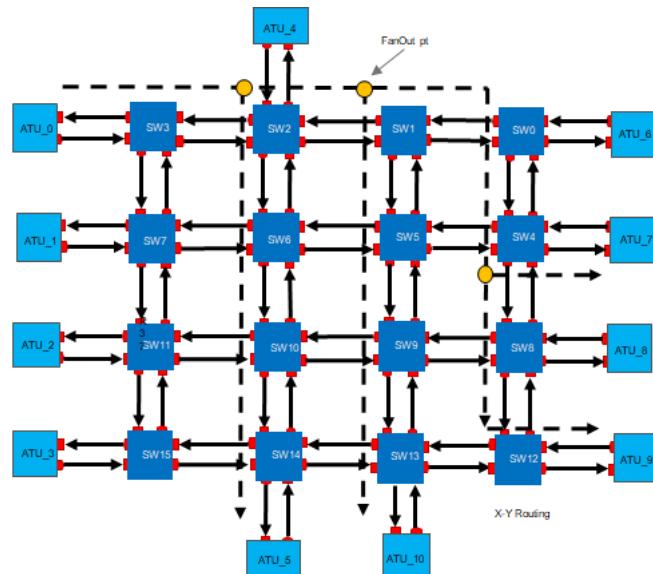
These routing algorithms are explained in the sections below. In Algorithm 1, described below, multicast trees are created for any 1-to-N combination and multicast nodes placement is not fixed but is rather determined solely by the multicast tree creation algorithm. In Algorithm 2, multicast nodes are placed at strategic locations in the network, for example, a multicast node can be placed in each quadrant or sub-quadrant of a mesh for broadcasting/multicasting from that node to the nodes in the sub-quadrant. Lastly, algorithm 3, discusses multicast for Ring networks. In ring networks, no specific multicast FanOut node is needed.

1. Algorithm 1:
  - a. Map each flow separately using the unicast routing algorithm.
  - b. Once all the multicast flows are routed, start to merge the flows to form multicast trees.
  - c. Merge Process: 1: Start from the leaf nodes and traverse the nodes in the path in the reverse direction. 2: When two or more paths converge at a node, designate that node as the FanOut node. 3: Multicast is enabled at the FanOut nodes.
  - d. Use Turn based routing with probability based route diversity for multicast routing.
  - e. The process of determining routing and merging is shown in Figure 24-1 and Figure 24-2.

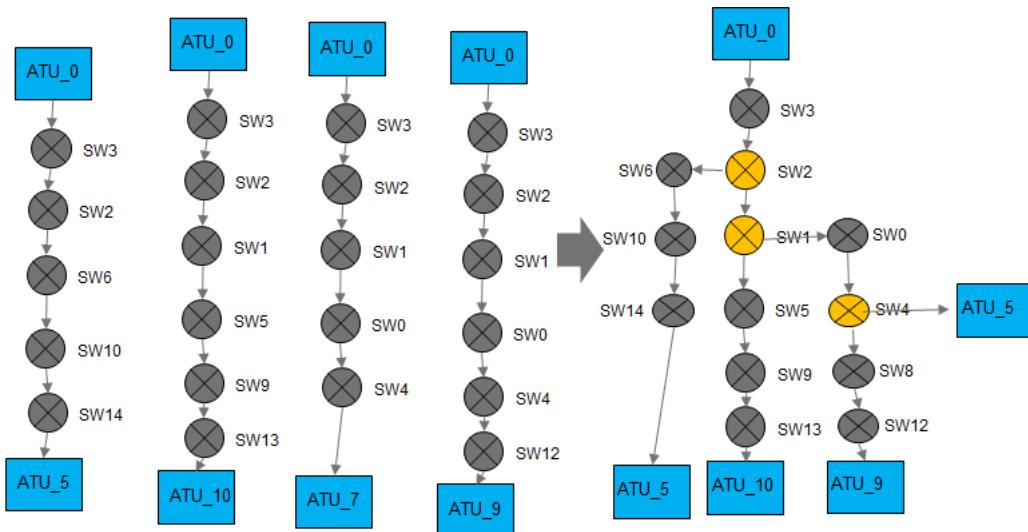
2. Algorithm 2: Quadrant based multicast
  - a. Setup multicast enabled nodes at Strategic locations within the network.
  - b. Between source (Initiator) and multicast relay and leaf nodes, send only one packet (unicast) per branch.
  - c. Once leaf multicast node is reached, unicast to individual destinations (Targets).
  - d. This algorithm is shown in Figure 24-3

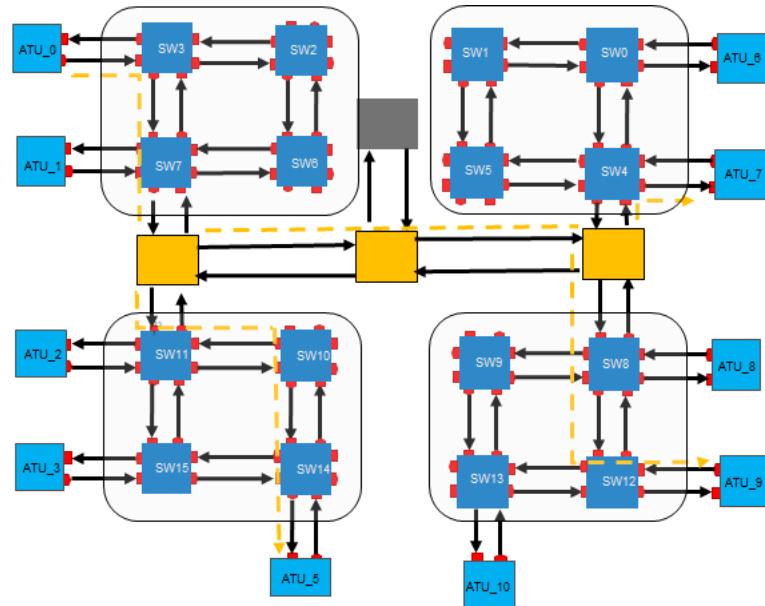
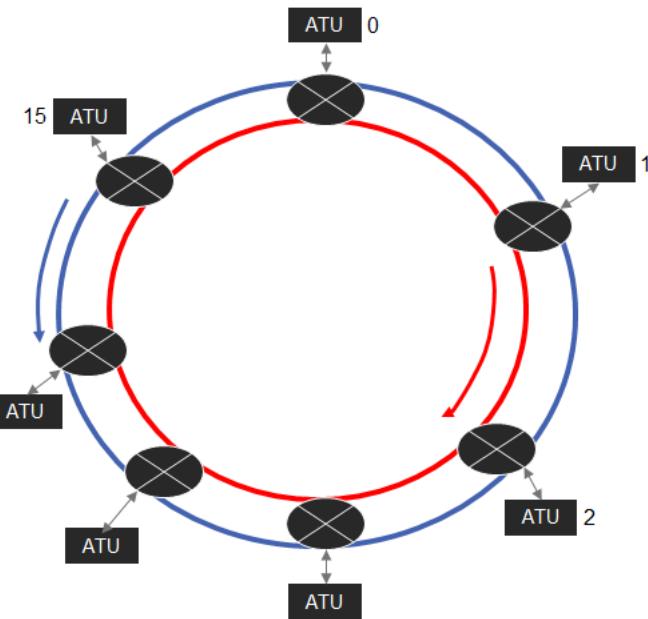
### 3. Algorithm 3: Ring Based Networks

- a. In a Ring network, every switch on the ring has a two input and two output ports. One input-output pair connects the switch to the upstream and downstream switches and the second pair connects the ring with an ATU connected to the Master/Slave pair as shown in Figure 24-4.
- b. When a multicast packet arrives at a switch, the multicast label is looked up. If the packet has to fanout at the switch, it is simply broadcasted to both output ports.
- c. Another option, could be to broadcast the packets at every switch, and let the Target ATU drop the packet if the packet is not meant for it.



**Figure 24-1 Example mesh network for multicast routing**



**Figure 24-2 Creating a multicast tree using unicast routing for individual flows****Figure 24-3 Quadrant based multicast routing.****Figure 24-4 Path based multicast routing in the Ring network.**

## 24.3 Deadlock Avoidance

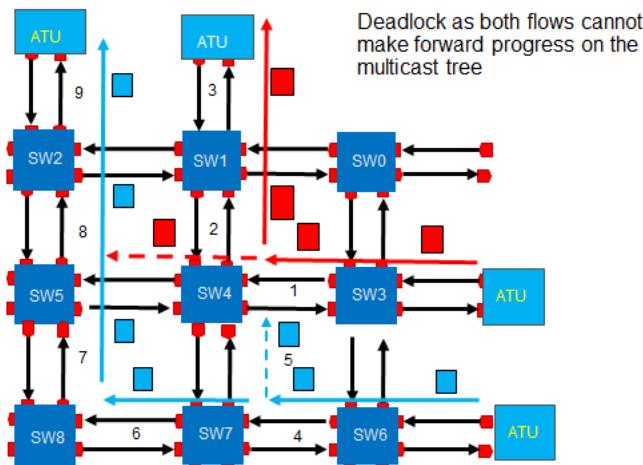
One of the main issues with multicast flows is deadlock. The deadlock condition for multicast flows can be defined as follows:

1. Two or more multicast packets need to share a network resource during their transmission in two or more directions.
2. Different multicast packets occupy the shared resources till the end of their transmission.

Consider the network shown in Figure 24-5, where two multicast flows, Blue and Red, target the same destination nodes and hence have to share the same resources. The Red flow occupies links 1, 2, 3 while the Blue flow occupies links 4, 6, 7, 8 and 9. Both flows will deadlock if the packets are multi-Flit packets and occupy the whole path.

The condition would happen even if the individual flows used deadlock free routing. Having virtual channels, would not solve this deadlock problem.

The only way to solve this deadlock issue to allow for store and forward operation at nodes where FanOut is required.



**Figure 24-5 The red and blue flows deadlock as they occupy different resources preventing forward progress**

## 24.4 Multicast Label and Routing Bits

Multicast label serves the purpose of identifying a particular multicast tree. The multicast label (ML) uniquely identifies a multicast tree in the whole system, and a multicast packet is uniquely identified by the tuple: {Transaction ID, Sequence number, Multicast Label}.

Multicast label provides an abstract way of defining the multicast tree and allows the system a handle on how to store and retrieve the information related to

Routing bits and multicast labels shall be simultaneously used in certain network configurations and will also be a substitute of each other in other network configurations. That is, multicast label can also be used to route packets.

In Figure 24-6 below, illustrates the packet with the Multicast label and the routing bits.



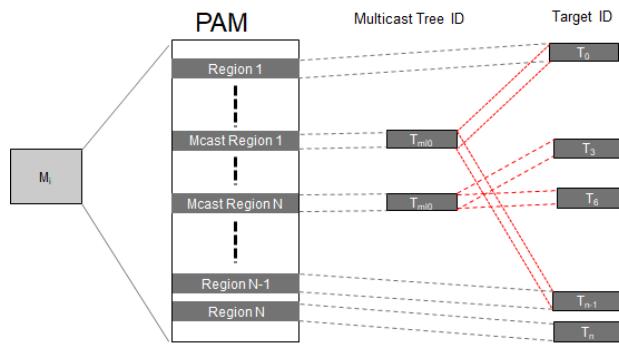
**Figure 24-6 Packet format with the multicast label**

As mentioned above, for configurations where multicast fanout is done in the switches, multicast label would be used to route multicast packets. The switch would perform a look on the multicast label to determine the fanout at the switch. Since unicast packets will also be routed through the switch, and the packet format for the Fabric is fixed, unicast packets will also carry the multicast label. The multicast label have a specific bit pattern to identify packets as unicast.

## 24.5 Partial Address Map

As explained in the previous section, each multicast tree is represented by a multicast label. A multicast label in turn references a tree representing all the slaves that are a member of that multicast tree. The master identifies the multicast tree much the same way it identifies any other slave in the system, by way of assigning an address region in the PAM table as shown in Figure 24-7 below. The PAM table also contains the attributes of the Target such as supported burst sizes and burst types. Care should be taken when the attributes of the multicast tree are defined. All members of a multicast tree should have the same attributes.

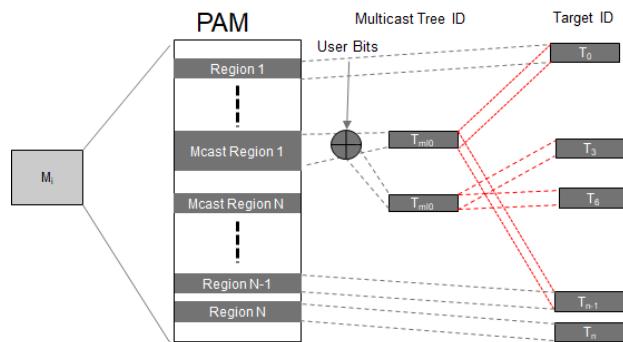
In the same manner the



**Figure 24-7 Multicast Tree addressing in the PAM Table.**

————— Note —————  
The customer will be asked if they want to setup a multicast tree. They will also be asked to pick a region in the PAM table that has not been assigned, for multicast labels. The customer then would be asked for each multicast region which Slaves are part of that multicast tree region. Symphony will assign a Multicast label. All transactions sent by the Master to the I-ATU that fall in the multicast address region will be multicasted to slaves represented by that multicast region.

Symphony also supports the use of “user bits”, should the customer use them to further define the multicast tree. For example, the customer has setup a broadcast tree, but wants to refine which nodes receive the current transaction by using “user bits”. Symphony enables that facility as shown in Figure 24-8 below.



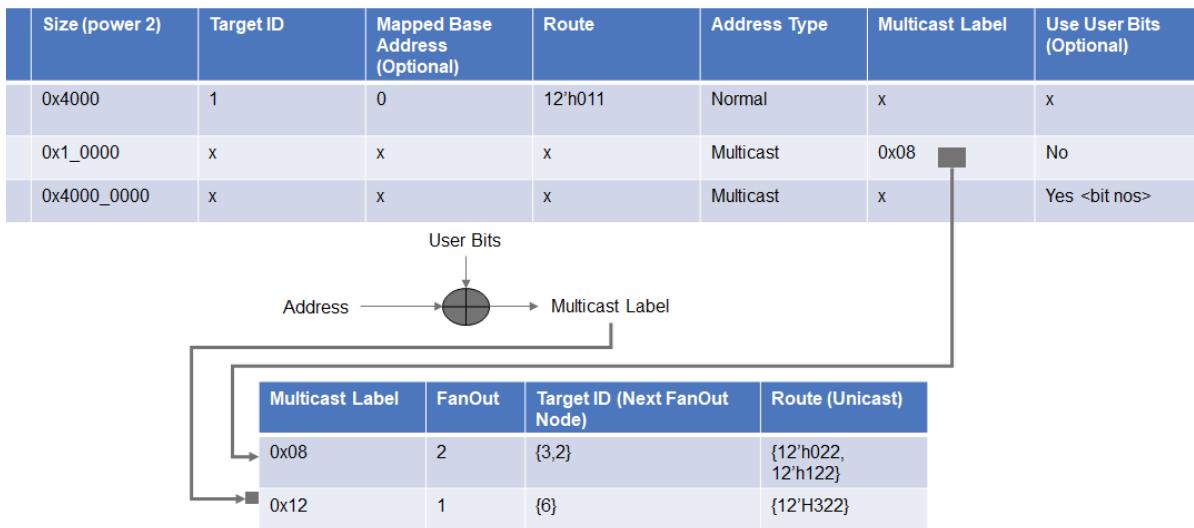
**Figure 24-8 Multicast Label modulated by Mcast Region and the User Bits**

If user bits are to be used to refine the multicast tree already setup, Symphony shall support the following configurations:

1. User bits can be used to index into the PAM table along with the multicast address region to identify the multicast tree and corresponding label. For example, the customer can setup a Multicast region to represent a multicast tree. The customer may then use user bits to define a sub-tree within the multicast tree along with the multicast transaction. User bits not carried with the transaction.
  2. User bits used as a mask to identify multicast ports in the multicast adapter. In this case the user bits are carried with the multicast transaction. For example, a broadcast tree label is setup corresponding to the multicast region used for broadcasting transaction by the application. The application then can use User Bits to identify which nodes within the multicast tree should the broadcast go to. Since the user bits identify the fanout ports at the ATU and the multicast adapters, they will be configured to use the user bits to identify fanout ports. In case the multicast network is hierarchical, multiple sets of user bit masks for each hierarchy would have to be provided by the user. Note that, if this approach is used, then the User has to be aware of the network topology.

The PAM table at the I-ATU will contain among other fields the following (for complete description of what will be contained in the PAM table, please see the Memory Map chapter). These additional fields needed to identify the multicast transaction are: the multicast label to use, Fanout, User of use bits, Target ID of the next fanout node (could be multiple nodes), Route bits (to get to the next Fanout node). Figure 24-9, shows the PAM table setup in the I-ATU.

It should be noted that the use of User Bits, forces the application to have the knowledge of the topology of the network, because the user bits signify how the packet will fanout in the network at different Multi-cast nodes/adapters.



**Figure 24-9 PAM table setup for Multicast**

## 24.6 Initiator ATU Multicast Support

The initiator ATU shall support the following:

1. Ordering requirements:
  - a. Ordering based on Channel ID and Ordering ID – Transactions with the same channel ID are ordered; Transactions with the same ordering ID are ordered with respect to each other. I-ATU will use information about the ordering model.
  - b. Recommended that Multicast ordering ID/Channel ID be different from Unicast ordering ID/Channel ID because of performance impact.
2. For buffered writes, the I-ATU will respond with a response, however, if there is an outstanding non-buffered write, the buffered write response will not be issued until the non-buffered write is completed:
  - a. The buffered write response from the Target would be discarded once received and the corresponding context cleared.
3. I-ATU will perform all target aware request splitting:
  - a. Boundary based (like AHB – 1KB boundary).
  - b. Size based (limit to max burst that target can accept).
  - c. Protocol capability based – split if target does not support wrap or has smaller burst support.

## 24.7 Target ATU Multicast Support

The Target ATU shall support the following:

1. Each target ATU checks the multicast label of the packet to ensure that the T-ATU is part of the multicast tree.
2. Context table stores the multicast label.
3. Response packet contains the multicast label.

The rest of the functionality of the T-ATU remains the same.

## 24.8 Multicast support in the Fabric

Symphony will offer the following solutions for multicast traffic within the Fabric:

1. Standalone multicast adapter which contains the FanOut as well as the FanIn adapter.
2. Buffered multicast enabled VC switches which is capable of Fanning out and Fanning In.

Both of these solutions will be described in the following sections.

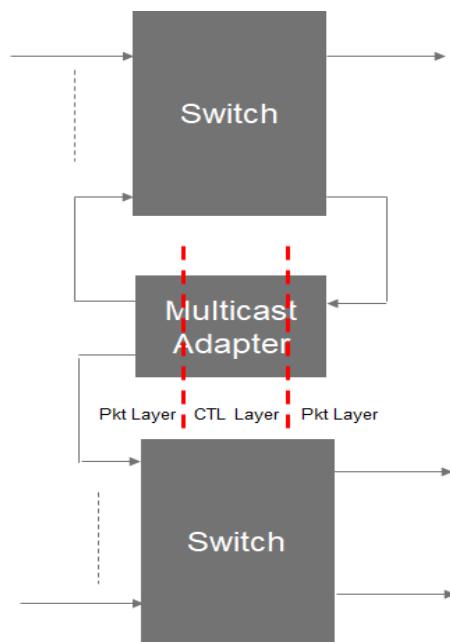
### 24.8.1 Standalone FanOut Multicast Adapter

The multicast adapter is placed at strategic locations in the Fabric. The multicast adapter is a store and forward node, in that, it stores the whole packet before it starts forwarding it via the output ports. Buffering the whole packet prevents deadlocks from happening when there are multiple independent multicast trees setup in the Fabric. The buffer cannot be cut through as that may cause multiple link resources to be occupied by the packet at the Fanout point and can cause deadlocks.

The packets are routed to the multicast adapter through the fabric using the routing bits in the packet header. As mentioned earlier, the packet contains both the routing field and the multicast label. The switches do not look at the multicast label. Figure 24-10 shows the multicast adapter attached to two switches. The multicast adapter is composed of the packet and the CTL layers as shown in Figure 24-10.

The Multicast adapter performs the following:

- ▶ Buffer the multicast packet.
- ▶ Look up the multicast label and determine how and where to forward the packet.
- ▶ Swap the old routing header with the new routing header for packets.
- ▶ Unicast the packets if the multicast packet is exiting from a single output port.
- ▶ Broadcast/multicast the packet to all the forwarding ports on the multicast adapter if needed/required.
- ▶ Each multicast adapter is capable of sending the buffered response back to the I-ATU.



**Figure 24-10 Multicast adapter connected two switches**

### 24.8.1.1 Multicast FanOut State Table

The multicast packet, as mentioned before, is uniquely identified by the tuple: {Transaction ID, Sequence number, Multicast Label}. The routing or forwarding of the packet can be defined by any of the following:

1. Fanout and routing identified solely by the Multicast label. That is, the state table is configured apriori, with the information of where to forward the packet based on the multicast label.
2. Fanout and routing identified by the multicast label and the user bits. The user bits act as a mask to prevent or allow multicast on the output ports in the multicast adapter.

In addition to the multicast label and the user bit usage indication, other fields that are kept in the state table are the:

1. Transaction ID of the packet
2. Sequence number
3. No of FanOut
4. Forwarding route (multiple route fields for each unicast that has to be performed per port)
5. Field to indicate whether user bits are used to identify the output ports and their locations.
6. Whether the packet has been responded to in the case of buffered write.
7. Error field to indicate that some sort of error happened on this transaction.

In Figure 24-11, the state table required by the FanOut adapter is shown. Note that each entry in the sub-table corresponds to one and only one transaction. If however, no response aggregation is done then the sub-table is not needed.

Multi-Label	FanOut	User Bits	Port 0 (Unicast Routing Bits)	Port 1	Port 2	Buffered Response	MsgID	Seq No.
011010	4	No	011010, 011011	011110	111011	Yes	11	0
110110	2	Yes <12,13,14>	011110	111000		No		1

Figure 24-11 Multicast Fanout table at the FanOut Adapter

### 24.8.1.2 Flow Control

The multicast FanOut adapter flow controls the upstream switches in much the same way the switches and the other network elements flow control each other.

The flow control is per port and per VC using a credit/ready and valid signals.

Other than the buffer occupancy, the multicast adapter will also flow control if the context table is full.

### 24.8.1.3 Multicast FanOut Adapter Packet Forwarding

Packet forwarding for a single input single output port FanOut adapter is shown in Figure 24-12. Packets arriving from the switch are buffered, their context looked up and forwarded. The Adapter has per-Vc buffers and each buffer stores the whole packet.

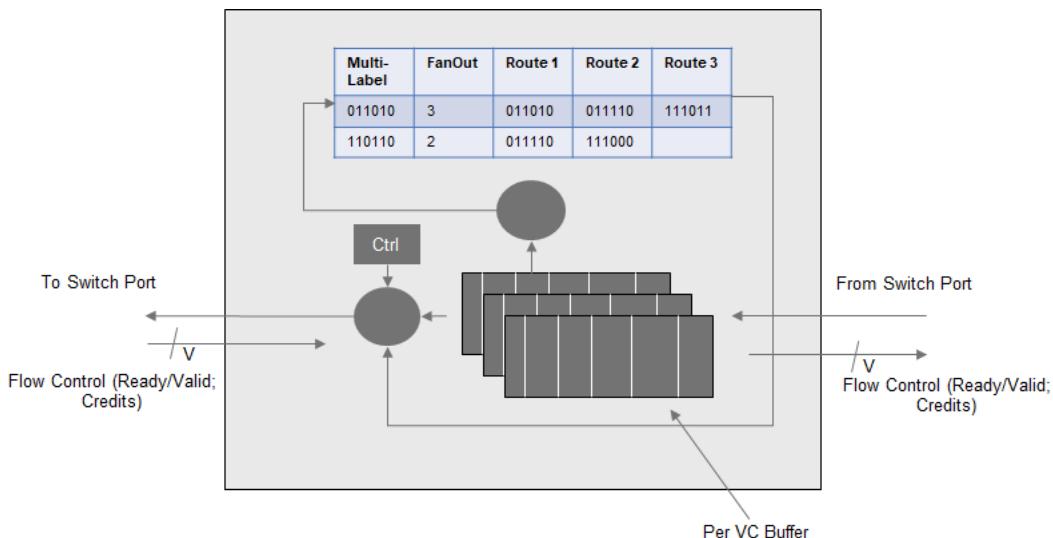
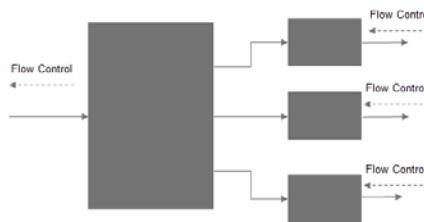


Figure 24-12 Single input single output FanOut Adapter

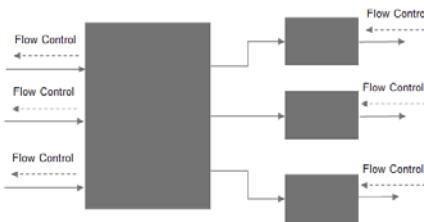
Packet forwarding for a single input multiple output ports is show in Figure 24-13. The packet is buffered at the input and broadcasted out to the output port buffers. Since each output port is independently flow controlled, the buffers may not all be empty at the time a packet arrives in the input buffer. The

adapter will broadcast to the output ports independent of the state of other ports. Details of the internal structure and buffer construction will be covered in the uArch spec.



**Figure 24-13 Packet Forwarding for Single Input multiple output Multicast FanOut Adapter**

For multiple input and outputs the internal architecture of the FanOut adapter is more complex as shown in Figure 24-14.



**Figure 24-14 Packet Forwarding for multiple input-output FanOut adapter**

## 24.8.2 Multicast FanIn Adapter

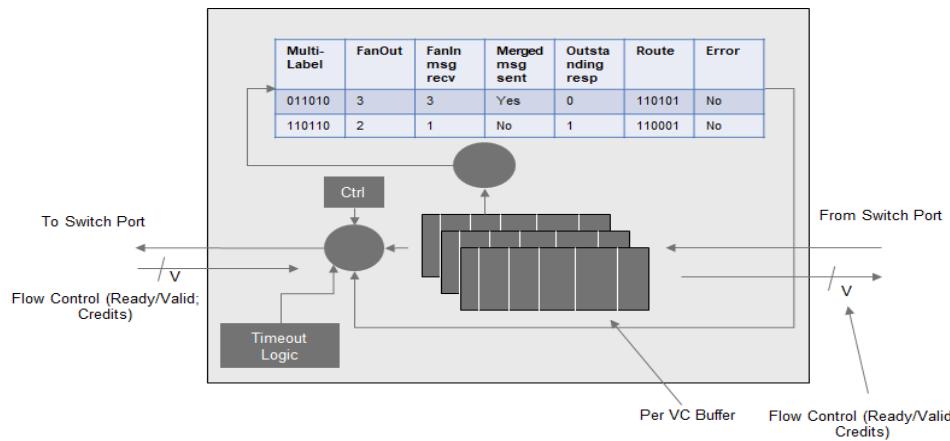
Standard protocols such as AXI, OCP, AHB, and APB are based on request-response semantics. That is, every request has a corresponding response.

When a multicast transaction is sent in any of the above mentioned protocols, each of the slaves that receive the multicast packet, will respond to it. If this response is not aggregated along the way, the I-ATU will receive just as many responses as there are leafs in the multicast tree. The 'n' responses traveling back are a cause of congestion and performance degradation.

Symphony aggregates responses in the FanIn adapter.

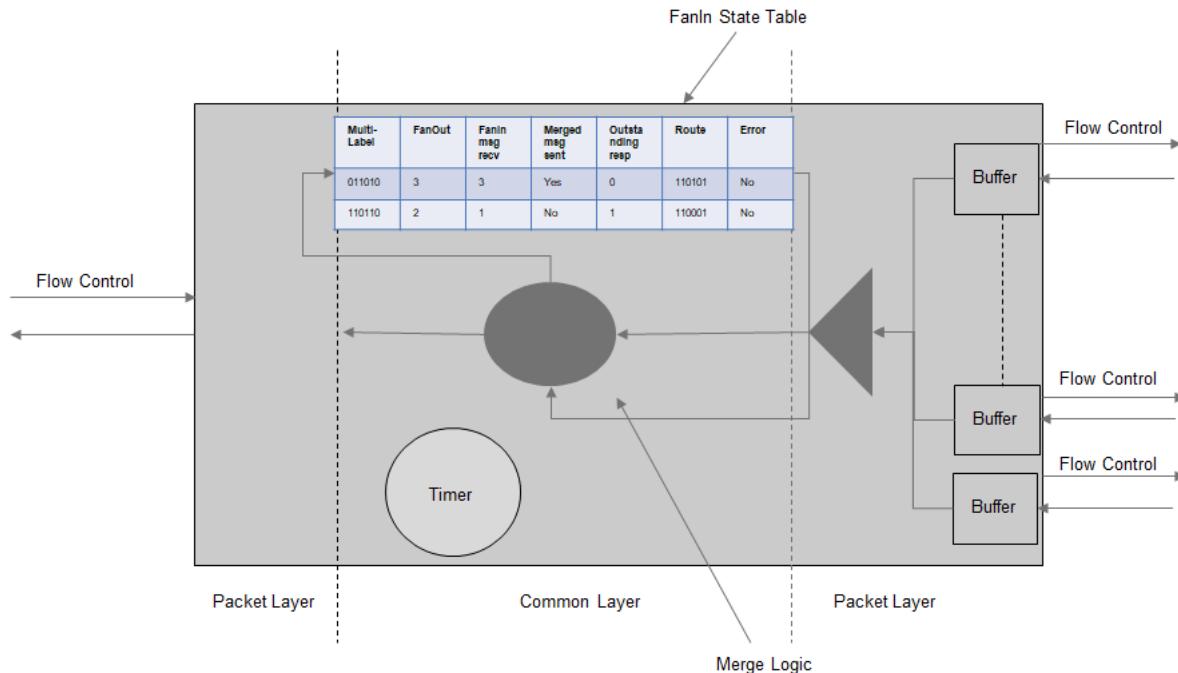
The Multicast FanIn adapter receives response which can include acknowledgments plus meta data or data. The FanIn adapter merges these responses and sends a single response upstream.

A simple FanIn adapter is shown in the Figure 24-15 below. The FanIn adapter is placed at the same location as the FanOut adapter so that both adapters can track and share state.

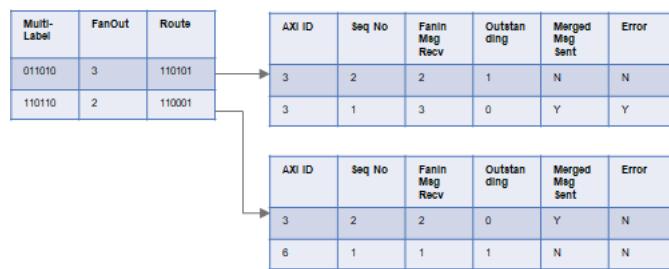


**Figure 24-15 Simple FanIn adapter**

Responses enter the FanIn adapter via the input ports as shown in Figure 24-16. The packet layer depacketizes the packet and sends the information to the CTL layer. At the CTL layer information pertaining to the packet header and meta data is stored along with any data if needed. The CTL layer also contains the context corresponding to the multicast label, the MsgID and the SeqNu. Figure 24-17, below illustrates the context information contained in the CTL layer.



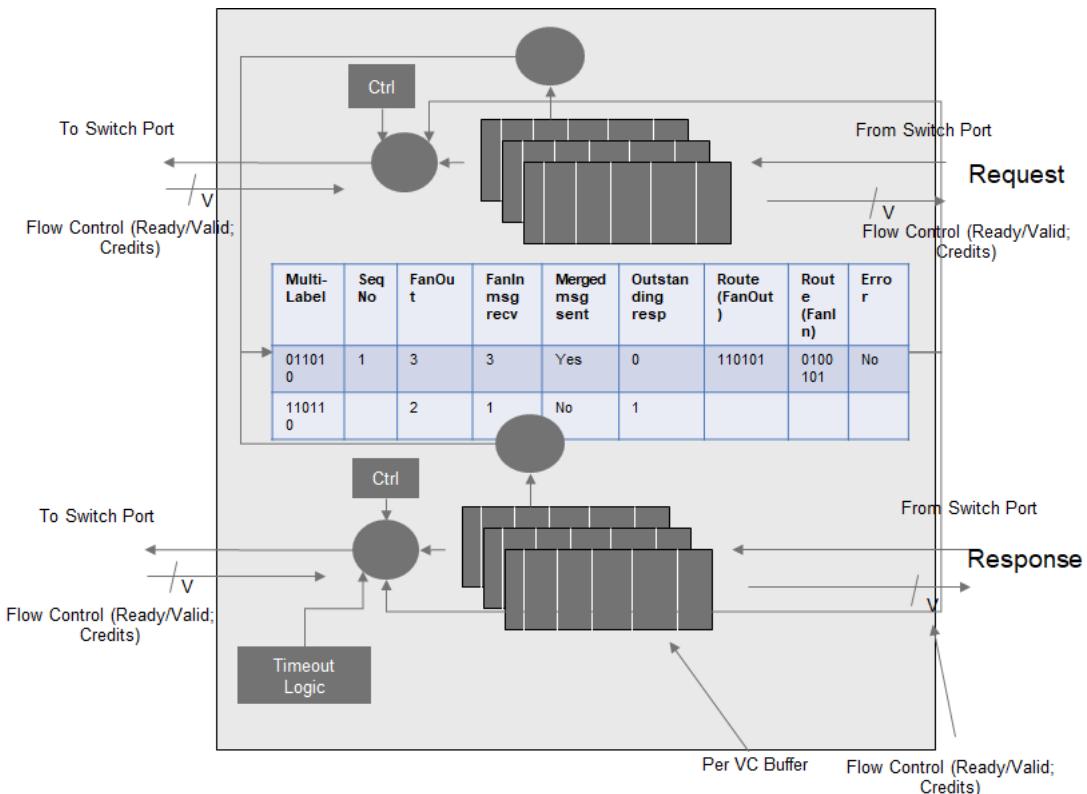
**Figure 24-16 Details of the FanIn Adapter**



**Figure 24-17 Packet and Multicast Label context stored in the FanOut Adapter**

## 24.9 Combining FanOut and FanIn Adapter - Multicast Adapter

The FanIn and the FanOut adapter is combined in what we call as the Multicast Adapter. The multicast adapter is shown Figure 24-18.



**Figure 24-18 Multicast Adapter - FanIn and FanOut**

## 24.10 System Level Considerations

The following system considerations need to be taken for supporting multicast traffic in the network:

1. In order for multicast traffic not to interfere with unicast traffic, it should always be carried on a dedicated VC.
2. It is important that the I-ATU split the transactions to the packet size that is supported by the Fanout Adapter. The Fanout Adapter supported packet size should be captured in PAM table at the I-ATUs.
3. The Fanout adapter will back pressure transactions, if there is not enough context space available for to store context for multicast transactions.

For buffered writes, the I-ATU will respond to the transaction. However, the slave will still transmit a response back. The FanIn adapters, will continue to merge and send responses back as they are normally configured to do. It would be the job of the I-ATU to squash/drop all the responses that come back for the buffered transaction that has already been acknowledged.

For posted writes, the master is not expecting any response to come back and neither the Slave is expected to send any responses back. The T-ATU will generate a response which will then go through the same merging process as any other response. I-ATU will squash all responses to posted write transactions.

## 24.11 Error Management

Error management on the multicast adapter includes the following:

### 24.11.1 FanOut error:

1. Multicast label not in the context table – send packet to the designated node in the Fabric
2. Multicast fanout attempts to send the packet out a port that does not exist – error notified by an interrupt
3. Part of the multicast tree is powered/clocked off
  - a. Power/clock/path disconnect error handled the same way as normal unicast error
  - b. Error packet routed back to the multicast FanIn node
  - c. FanIn node updates the Multicast label error state, so that node does not wait for responses to come back from the errored paths.

### 24.11.2 FanIn Error:

1. Multicast label not in the context table – send packet to the designated node in the Fabric
2. Multicast fanIn attempts to send the packet out a port that does not exist – error notified by an interrupt
3. If the timeout counter expires:

- a. If no responses have been received, send error message to the Initiator
- b. If some responses have been received, merge and send them to the Initiator with an error indication

Note that normally it would not be expected that, for example a packet arrives at the multicast adapter and its multicast label does not match any label in the adapter. However, such a condition can happen, if bit errors are introduced during transmission of the packet. Hence situations like these need to be handled explicitly in the architecture.

## 24.12 ARTG and ADM

(Major part of this section are from the discussion Federico, Youcef and I had and from his slides that were based on the discussion).

ADM defines an entity called Multiflow. Multiflow is defined as:  $\text{Socket} \rightarrow \{\text{Socket}_i, \text{Socket}_j, \dots\}$ . The Maestro flow is shown below in

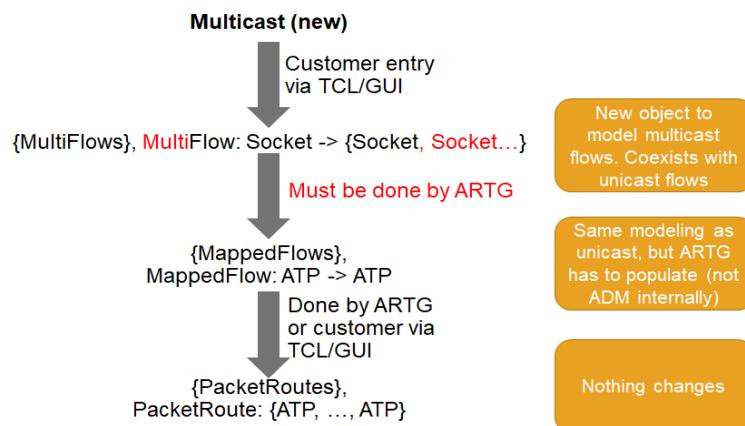


Figure 24-19 Maestro Flow for Multi-flow flows.

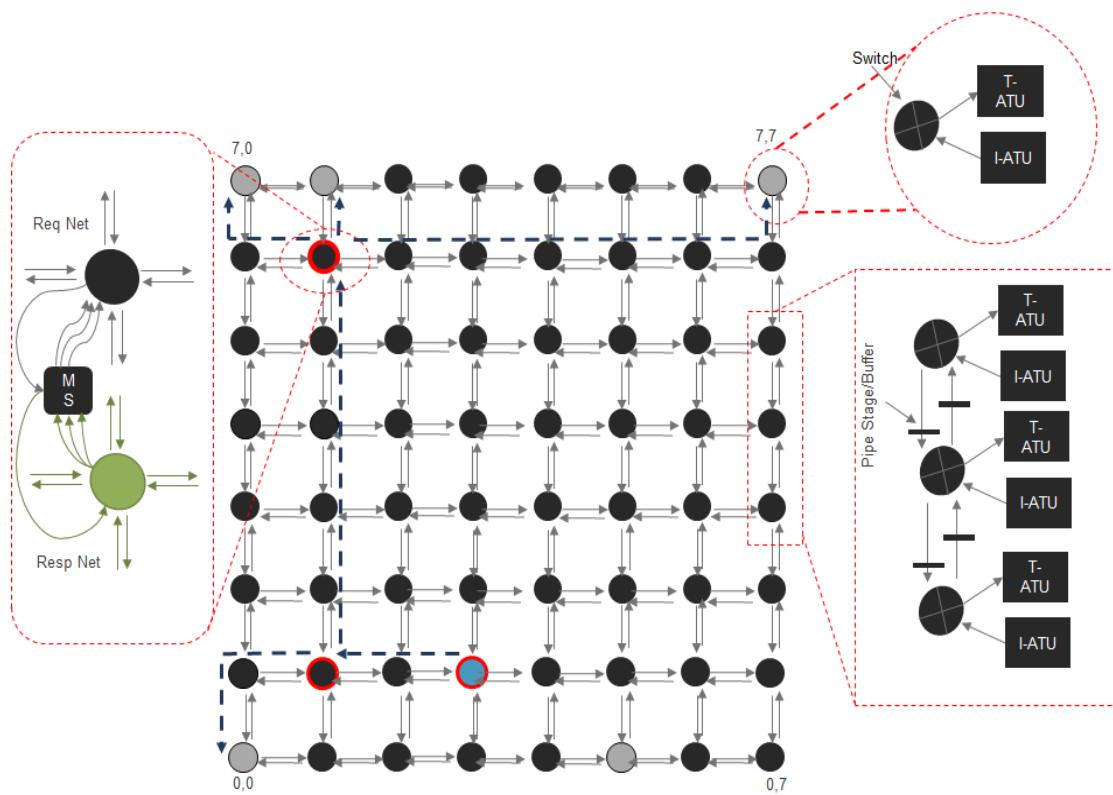
Customers enter the details of the multi-flows and they can also enter the location of the multicast stations they want and where they would like to locate them. The routes from the source to the multicast stations and from the multicast stations to the target nodes are either provided by the user or determined by Maestro.

In the case, where ARTG (Automatic Regular Topology Generation) is invoked by Maestro, the ARTG algorithm multicast extension will determine the placement of the multicast stations, the path that the packets will take from the source to multicast stations and from multicast stations to destinations and configure the I-ATU, the T-ATU and the Multicast stations.

### 24.12.1 Configuring Network Elements for Multicast

Symphony allows the system designer some options on how to configure multicast in the system. As was mentioned before, “Multicast Labels” are unique in the system. The options are: 1) Maestro automatically generates Multicast Labels; or 2) the user assigns a particular label to the multicast tree.

Once a multicast tree has been setup the user can choose to multicast to all destinations in the tree or to only a subset of the destinations by sending in a multicast mask using “user bits”. If no user bits are configured as the multicast mask, Symphony will multicast to the whole tree. Consider the following example shown in Figure 24-20 below:



**Figure 24-20 Example 8x8 mesh network Src:(1,3) ==> Dest{(0,5), (0,0), (7,0), (7,1), (7,7)}. Multicast Stations: MC0:(1,3), MC1:(1,1), MC2:(6,1).**

The 8x8 mesh network above has an Initiator ATU and a Target ATU attached to each switch. There is one multicast tree in the system. The Source node (1,3) is multicasting to a set of destinations  $\{(0,5), (0,0), (7,0), (7,1), (7,7)\}$ . Symphony provides a facility (ARTG and Multicast Insertion) which will automatically generate the mesh and if provided with the multicast flow, will insert the multicast stations in the optimum places. In the above example, the multicast insertion algorithm inserts multicast stations at nodes (1,1) and (6,1). The inset figure shows how the multicast station is connected with the switch in the request and the response network. The multicast station fans-out the requests and aggregates the responses.

There are three components that need to be configured to enable multicast. These are the I-ATU, T-ATU and the Multicast Station. Using the example above, the configuration of the I-ATU (1,3) will be as follows:

- ▶ PAM Table Setup: Multicast transactions are identified by the I-ATU by the target address that comes with the transaction on the AW channel. The PAM table entry is configured the same way as any other PAM table entry, except that the entry points to a multicast label, the ID of the next multicast station and the route to the next multicast station. The user can also specify “user bits” to determine which targets in the multicast tree should receive the transaction. The user also has to describe which user bits represent which targets. The figure below shows how the I-ATU (1,3) would be configured.

<pre> ATU-I: pamMulti = true          /*Indicates that Multicast is supported pamMultiLabel = 1         /*Label provided by Maestro. Label                            No.1 pamMultiUser = false      /*Label not provided by the User pamMultiLabelUserBits = X /*User Bits that define User Label pamMultiMaskUserBits = [4:0]/*User Mask bits that identify targets                            that should receive the transaction </pre>
<pre> ATU-I PAM pamBaseAddress = h80000000 pamBaseMask = hFFFFFFF00 pamSz = 64B </pre>
<pre> ATU-T {(0,0),(7,0),(7,1),(7,7),(0,5)}: pathLut =[{targ_id: {label 1; Id of the Src}; route: route to MC2},            {targ_id: {label 0; Id of the unicast Src}; route: route to unicast Src}] </pre>

**Figure 24-21 Configuration of the I-ATU/T-ATU**

- Multicast Station Setup: The multicast station configuration has multiple fields that are configured. Symphony will configure them automatically if ARTG is used. Otherwise if the user wants to setup the multicast stations manually then the fields in the figure below would need to be configured.

```

Multicast Tree:
Src (1,3) → {(0,0),(7,0),(7,1),(7,7),(0,5)} → MC Label 1

Multicast Station 0:
mask2TRGrp = [00001, 11110] → Mask per output port for the MC
maskField   = 'H_multi_mask[4:0]
trgPathLut  = [{targ_ID:Id of (0,5), route:' },{targ_ID: Id of (MC1), route:
to_MC1}]
trgPathEgress = [{targ_ID:Id of (0,5), egress:0},{targ_ID: Id to (MC1), egress:1}]
SrcPathLut = [{targ_ID: Id of the Src; route back Src}

Multicast Station 1:
mask2TRGrp = [10000, 01110] → Mask per output port for the MC
maskField   = 'H_multi_mask[4:0]
trgPathLut  = [{targ_ID:Id of (0,0), route:'010},{targ_ID: Id of (MC2), route:
to_MC2}]
trgPathEgress = [{targ_ID:Id of (0,0), egress:0},{targ_ID: Id to (MC2), egress:1}]
SrcPathLut = [{targ_ID: Id of the Src; route back MC0}]

Multicast Station 2:
mask2TRGrp = [01000, 00100, 00010]
maskField   = 'H_multi_mask[4:0]
trgPathLut  = [{targ_ID:Id of (7,0), route:to_(7,0)}, {targ_ID: ID of (7,1),
route_to_(7,1)}, {targ_ID: Id of (7,7), route_to_(7,7)}
trgPathEgress= [{targ_ID:(7,0), egress:0}, {targ_ID:(7,1), egress:1},
{targ_ID:(7,7).egress:2}]
SrcPathLut = [{targ_ID: Id of the Src; route back MC1}

```

**Figure 24-22 Configuration of the Multicast Stations**