ATM CellSwitch Application Guide

CSTAATMCS-UG/D

Draft

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

1.1 Purpose of Document

This document presents the software architecture and design for Motorola's Asynchronous Transfer Mode (ATM) Cell Switch reference application targeted for the combined C-5e network processor and Q-5 traffic management co-processor.

1.2 Software Architecture Overview

The C-5e network processor is divided into four clusters of four channel processors, where each cluster is typically used to provide a common function across all four processors. For this application only two clusters are used. The cluster usage is shown in Figure 1.

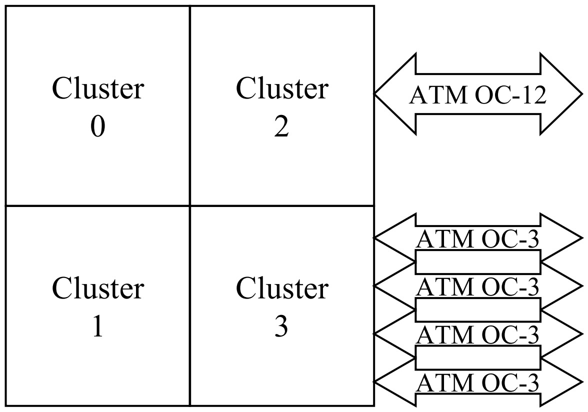


Figure 1. Cluster Usage

1.3 Feature Overview

This application demonstrates the following features

- Support for full-range Virtual Path Identifier(VPI)/Virtual Channel Identifier(VCI) ATM cell switching at both OC-3 and OC-12 rates
- Recognition of Operations Administration and Maintenance (OAM) and Resource Management (RM) cells and routing of those cells to the eXecutive Processor Reduced Instruction Set Computer(RISC) Core(XPRC)
- Cluster aggregation in support of OC-12
- Support for the Combo-2 Physical Interface Module
- Synchronous Optical Network(SONET)/Synchronous Digital Hierarchy(SDH) support
- Support for ATM Constant Bit Rate(CBR) and Variable Bit Rate(VBR) policing
- Support for per customer bandwidth allocation
- Instrumentation for performance analysis

- Host configuration of the Q-5 traffic management co-processor

In addition to the clusters of channel processors, there is an XPRC and a fabric processor (FP) present on the network processor. The FP in this reference application is configured for C-5e Back-To-Back mode.

The XPRC processing in this application performs the following functions;

- Initialization and configuration of the network processor
- Print servicing for textual I/O from the channel processors
- SONET Monitor alarm processing for alarms reported from the channel processors
- Dequeue and discard of discard traffic discarded by the Q-5 policing algorithms
- Dequeue and discard of OAM traffic. Eventually this traffic might be routed to a host program.

1.4 Applications Component Usage

This application relies on the following applications components;

- Phy – Common bit level microcode
- SonetQ – Q-5 version of Common SONET Monitor processing
- TableUtils – Common offline table restoration processing
- CommSvc – Common Host coordination processing
- Fabrics – Common microcode for the various fabric modes
- Performance – Common defines for instrumenting for performance analysis

See the documentation in apps\components\<componentName>\doc directory for the documentation on the applications components that this application uses. One exception to this is the sonetQ component, please see the documentation in the sonet applications component since the processing is identical and sonetQ has only minor modifications to accommodate the use of the Q-5 instead of the Queue Management Unit (QMU) for enqueuing and dequeuing.

1.5 Data Flow Overview

There is a single main data flow through this application, ATM cell switching. The processing involved does not really differ much based on the ingress or egress port being either OC-3 or OC-12. A step-by-step breakdown of the data flow is shown in Figure 2. The processing is essentially the same when one of the ports involved is the FP. Each of the descriptor formats is fully defined in the Buffer Management Unit (BMU) section later in the document. Full details on lookup keys and response data are provided in the Table Lookup Unit (TLU) section later in the document.

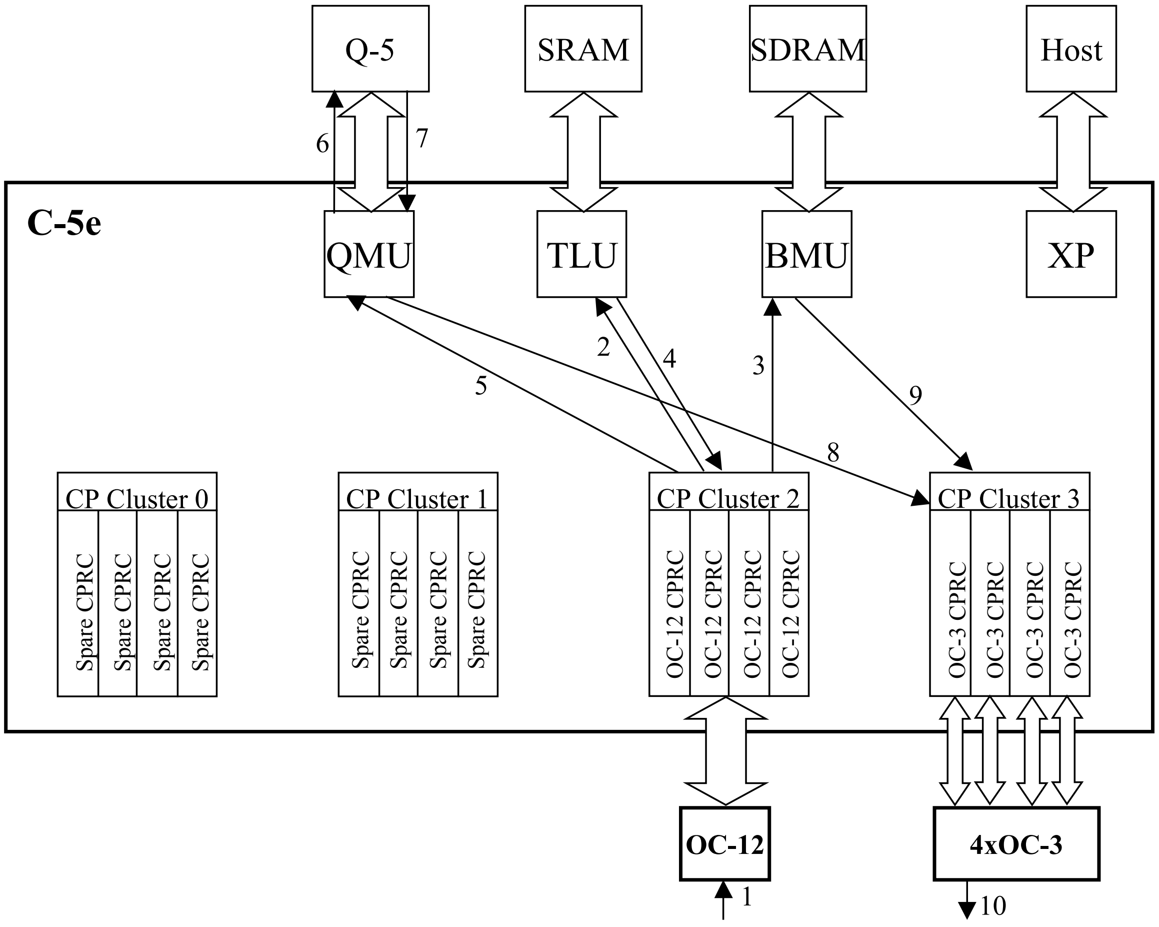


Figure 2. Step-By-Step Data Flow

1. ATM Cell is received at the OC-12 Port
2. Serial Data Processor (SDP) launches Lookup based VPI/VCI and ingress Port.
3. SDP/Channel Processor RISC Core (CPRC) Direct Memory Access's (DMAs) cell payload to BMU Buffer Memory
4. CPRC receives and processes lookup results from TLU
5. CPRC enqueues cell descriptor to QMU specifying a Traffic Queue (TQ) Id
6. QMU forwards descriptor to Q-5
7. Descriptor makes it's way through the scheduling process and is returned to the QMU to fulfill available Virtual Output Port (VOP) credit
8. CPRC dequeues the descriptor from the QMU specifying a VOP Id
9. SDP/CPRC DMAs cell payload from BMU Buffer Memory
10. SDP transmits ATM cell out one of the OC-3 ports providing a new cell header.

1.6 Application Configuration

There are a number of "magic numbers" for the application, such as the CPRC servicing each OC-3 port, the first TQ associated with a given port, the table Id for the Virtual Channel (VC) lookup table, etc. For the most part these numbers have been isolated to a single header file, appConfig.h. Changing any one of these values may not be as simple as changing it in this one file, although in some cases it is, but by isolating them here they are more easily changed, better defined, and also then appear mnemonically in the code which aids in code readability.

2 RELATED DOCUMENTS

- AF-TM-0121.000 Version 4.1, Traffic Management Specification, ATM Forum
- CSTCPHYC-UG, PHY Configuration Component Guide
- CSTCSMC-UG, SONET Monitoring Component Guide
- CSTCTLUC-UG, TLU Configuration Component Guide
- CSTCFPC-UG/D, Fabric Processor Configuration Component Guide
- C5EC3EARCH-RM/D, C-5e/C-3e Network Processor Architecture Guide
- Q-5 Traffic Management Coprocessor, FPGA Revision, Architecture Guide
- C-Ware Q-5 TMC API User Guide
- CSTOBSC-UG/D, Build System Conventions

3 Detailed Design

3.1 ATM Processing

The OC-12 port is served by an entire cluster of CPRCs. Each OC-3 is served by a single CPRC. With the exception of some token passing in the aggregated case of the OC-12 the processing is the same for both the OC-3 and OC-12. The processing for both will be described below, where they differ will be pointed out.

3.1.1 ATM Initialization

The DCPmain program, which serves as the entry point for ATM processing, is fairly simple. It initializes the various C-Ware Programming Interface (CPI) services which receive and transmit processing will rely on (kernel, buffer, pdu, table, and traffic management services), the VOP to be used by this CPRC's receive and transmit processing , establishes the buffer pool this CPRC will allocate from and initializes the pool, and registers a handler for QMU interrupts. For OC-3 it also disables token passing since the CPRCs operate independently. Both also set the aggregation mode appropriately for traffic management, 4 way aggregation for OC-12 and 1 way for OC-3. Processing then zeros out the statistics buffer and calls common ATM initialization processing.

ATM initialization processing simply allocates buffers to preload the two scopes for both the receive processing and the transmit processing.

The initialization processing then initializes the launch pad for the SDP lookups, creates the receive and transmit threads, and calls SONET initialization routines. For details on SONET Monitor processing, please see the component documentation for the shared SONET Monitor applications component. Once initialization completes, the main program calls ksContextExit releasing control to the receive and transmit threads.

3.1.2 ATM Receive Processing

The OC-12 and OC-3 processing share common microcode for most SDP components. All of the details provided regarding RxBit, RxSONET, RxSync, RxByte apply to both domains.

3.1.2.1 RxBIT

The RxBit processing concerns itself mainly with SONET framing and is responsible for monitoring Out-Of-Frame conditions and maintaining the current Loss-Of-Frame status. It plays no real role in the processing of individual cells. OC-3 and OC-12 have different microcode for this component but the differences are all associated with the fact that OC-3 is a 1 bit wide physical interface and OC-12 is a byte wide physical interface. The OC-3 processing therefore must run the bits from the physical interface through the shift registers in order to convert them into byte wide chunks for processing. Other than that, the processing is essentially the same.

3.1.2.2 RxSONET

The RxSonet block is a configurable, non-programmable, unit which processes the SONET framing and handles processing for all of the overhead bytes. It also plays no role in the processing of individual cells, beyond extracting them as a data stream from the SONET frames. It does however form the basis for our SONET monitoring implementation. For full details on our SONET monitoring processing, please see the documentation for that component in the apps/components/sonet/doc directory of your C-Ware Software Toolset (CST) installation.

3.1.2.3 RxSync

The RxSync processing provides cell delineation for the ATM interface. Accordingly, the processing for this component is organized around a 3-state state machine with states of Hunt, Pre-Sync, and Sync. Cells are only forwarded to the RxByte processing when the state machine is in the Sync state. While in the Sync state, this processing performs the Header Error Control (HEC) check for every cell and detects idle and unassigned cells based on VPI/VCI. For a non-idle, non-unassigned cell with a good HEC, this processing will also perform a Cyclical Redundancy Check (CRC) 10 check. No payload is forwarded for unassigned/idle cells or cells which fail the HEC check, merely the cell header and a HEC indication. The HEC byte is replaced with a status code used to convey this information. When the cell payload is forwarded, the payload is followed by a status byte indicating the success or failure of the CRC 10 check. The CRC 10 status is only of use when the cell ends up being an OAM cell. OC-3 and OC-12 use common microcode for this component.

3.1.2.4 ATM RxByte

The RxByte processing starts with waiting for the token. For OC-3, this is always a non-wait condition. For OC-12, for any given cell, only one of the four aggregated RxBytes will own the token and therefore, only that one will process the cell. The rest simply discard the bytes associated with the cell before checking again for token ownership.

Cell processing then starts with waiting for extract scope ownership. A congestion drop count is initialized to zero and scope ownership is checked. If the SDP does not own the extract scope, it must drop the incoming cell and continue to wait, keeping count of how many cells are dropped while waiting. Once the SDP owns the scope, the congestion count is written into the scope for use by the CPRC for maintaining statistics and header processing begins.

Header processing strips the cell header from the data stream, copies it to extract space, and forms a table lookup key from the VPI/VCI fields along with the CPRC Id taken from RxByte Control space. The processing then screens the VPI/VCI/Payload Type Indication (PTI) fields to detect OAM/RM cells and writes the results to extract space. The HEC byte is then checked. As noted above, the HEC is actually checked by the RxSync processing and the HEC byte is then re-used to indicate either a good HEC, a failed HEC, or an idle cell. If the cell is an idle cell, processing waits for the Merge9 indication from RxSync and restarts header processing with the next cell. If the cell has a failed HEC, this is indicated in extract space. Processing then passes the token, indicates header processing is complete, waits for the Merge9 indication from RxSync, switches scope, and returns to the start of processing waiting for the token again.

If the cell is a non-idle, good HEC cell, processing passes the token, indicates header processing is complete, and launches the lookup on the key constructed earlier. The cell payload is then streamed to the buffer via the DMA. The final byte after the cell payload is a CRC 10 status from RxSync. This byte is written to extract space, a done flag is then set and scope is switched. Processing then returns to the start, waiting for the token again.

3.1.2.5 ATM CPRC Receive Thread

The CPRC receive thread consists of a large While(1) loop which repeats the following steps endlessly. The processing begins with waiting for a header processing complete indication from the RxByte component. Processing then gets a handle to the extract space and update the statistic for cells dropped due to congestion. Cell processing then begins by checking the header status for the cell. If the cell failed the HEC check, the processing increments a statistic, waits for the cell payload to complete being written to the buffer, and simply sets up for receiving the next cell before switching scope and returning to waiting for header complete again. In the case of OC-12, the processing must also wait to get the enqueue token and pass it on before switching scope.

For a non-errored cell, processing now begins waiting for the results of the lookup launched by the RxByte component. If the lookup fails, the processing is identical to an errored header except a different statistic is incremented. For a successful lookup the processing then checks the valid connection bit. This is a bit in the table entry which allows the VC to be disabled but still leave the route in the table. If the VC has been disabled, the processing is identical to a lookup miss. If the VC is enabled, the processing begins building the descriptor for forwarding the cell. The processing then checks to see whether the cell is an OAM/RM cell or a normal user cell. For an OAM/RM cell, the descriptor is forwarded to the XPRC with the ingress cell header and the results of the CRC 10 check. For a non-OAM/RM cell, the route is then checked to see if it is out the fabric. If the route is out the fabric, the destination queue is set accordingly and the VC Index field is set to indicate the destination queue from the route. If the route is a non-fabric route, the destination queue is set to the destination queue from the route.

The forwarding logic from here on is the same for all three cases, the lookup is released, a buffer is allocated for the next cell to be received in this scope, and the processing begins waiting for the cell payload to complete reception. Once the payload is complete, for OC-12, the processing must wait to receive the enqueue token. The scope is then prepared for the next cell reception and the scope is released. The packet length is written to the descriptor and the enqueue is then performed. OC-12 then passes the token to the next processor and receive processing is complete.

3.1.3 ATM Transmit Processing

The OC-12 and OC-3 processing share common microcode for most SDP components. All of the details provided regarding TxByte, TxSONET, and TxBit apply to both domains.

3.1.3.1 ATM CPRC Transmit Thread

The CPRC transmit processing is another endless loop. For OC-12 this starts with waiting for the dequeue token. Once the CPRC owns the dequeue token it begins waiting for a packet to dequeue. For OC-3 the loop starts with waiting for a packet to dequeue. For OC-12, once the dequeue is complete, the dequeue token is passed to the next CPRC. Processing then begins waiting for a merge scope from the SDP. Once the merge scope is available it is filled out with the egress cell header and the transmit type. There are two transmit categories, switched cell and a variety of OAM/RM cell types. Transmit statistics are then updated and the last buffer transmitted by this scope is freed. This buffer is then recorded to be freed later. Processing then sets up the DMA action for transmission and releases the merge scope to the SDP.

3.1.3.2 TxByte

TxByte initialization establishes the Creg address to enable checking for merge scope ownership, asserts the TxBit signal, and turns off scrambling so that the cell header will be transmitted unscrambled. TxByte processing starts with waiting for a merge scope from the CPRC. Once the merge scope is available processing then sets up to read transmit type from merge space and waits for data valid to be indicated. The processing then reads the transmit type, sets up to read the cell header from merge space, initialized the HEC engine, and branches based on the value of transmit type. The two paths are switched cell and OAM/RM cell.

The switched cell path transmits the cell header calculating HEC as it goes, transmits the calculated HEC value, turns on scrambling, and then transmits 47 bytes of payload from the buffer. The final byte of payload is then transmitted with Merge9 asserted. The scope is then released, scrambling is turned off, and processing returns to waiting for the next scope.

The OAM/RM cell path transmits the cell header calculating HEC as it goes, transmits the calculated HEC value, turns on scrambling, initializes the CRC 10 engine, and then transmits 46 bytes of payload from the buffer calculating CRC 10 as it goes. 6 bits from the 47th byte of payload is then transmitted with 2 bits from the calculated CRC 10. The final 8 bits of the CRC 10 is then transmitted with Merge9 asserted. The scope is then released, scrambling is turned off, and processing returns to waiting for the next scope.

3.1.3.3 TxSonet

The TxSonet block is a configurable, non-programmable, unit which provides the SONET framing and handles insertion for all of the overhead bytes. It plays no role in the processing of individual cells beyond framing them as a data stream into SONET frames.

3.1.3.4 TxBit

The TxBit processing basically just streams bytes while monitoring for an out-of-frame condition signaled by RxBit. The out-of-frame condition causes TxBit to signal the TxSONET block when it needs to re-establish frame boundaries. In the case of OC-3, where the physical interface is only 1 bit wide, the processing must use the shift registers to transmit the bytes out one bit at a time.

3.1.4 Data Structures

There are two categories of data structures that should be noted, statistics and SDP interfaces. In all cases, the data structures used are identical for OC-3 and OC-12 and the material that follows applies to both.

3.1.4.1 Statistics

Each CPRC maintains a number of statistics in a statistics structure. To get the cumulative statistics for the OC-12 port, it is necessary to add up the individual statistics for each of the CPRCs servicing that port. The statistics structure is a collection of 9 counters. Each counter is a full 32 bit unsigned value and is initialized to zero as part of the CPRC initialization processing. The following table provides the field name for each of the statistics, in the order in which they appear in the structure, and a short description of what each counter is tracking.

Field Name	**Short Description**
rxGoodCells	Counts the number of user data cells received and forwarded
rxOamCells	Counts the number of OAM/RM cells received and forwarded
rxCongestDrops	Counts the number of cells dropped by the SDP because the CPRC was not ready to receive the next cell
rxHecErrored	Counts the number of cells received that were not forwarded due to HEC errors
rxInvalidVc	Counts the number of cells received that were not forwarded due to either a

	lookup failure or a disable route in the table
enqueueFail	Count the number of enqueue failure interrupts for descriptors with associated buffers (cells)
otherEnqueueFail	Count the number of enqueue failure interrupts for descriptors with no associated buffer (SONET reports)
rsvd1	Unused
txCells	Counts the number of user data cells transmitted
txOamCells	Counts the number of OAM/RM cells transmitted
rsvd2	Unused
rsvd3	Unused

Figure 3. Statistics

3.1.4.2 SDP Interfaces

There are three main data interfaces between the SDP and the CPRC, extract data which is passed from RxByte to the CPRC, merge data which is passed from the CPRC to TxByte, and the control data spaces which are typically used to communicate initialization information to the SDP components.

There are two copies of both the extract data and the merge data which allows the CPRCs and SDPs to be pipelined so that one copy can be being used by the CPRC while the other is being used by the SDP. These copies are typically referred to as scopes and the SDP can only see one copy at a time. Switching scope changes which copy the SDP sees. Each merge space and extract space has 64 bytes although applications often don't use the full space available. The tables below show the usage of the extract space and merge space for this application.

(12 bits) VPI (4 high order bits of GFC if UNI)		(16 bits) VCI	(4 bits) PTI/CLP
int8u pduHdrStatus	int8u congestionDrops	int8u crc10Indicator	int8u encodedPTI
int8u camValue	(24 bits) Unused		
(13 words) Unused			

Figure 4. Extract Space Usage

The extract space is used mainly to communicate error/non-error status of the cell and the type of cell. There are only two cell types recognized by this application, User Data and OAM/RM. The cell type is conveyed by the encodedPTI field. The extract space also contains the ingress cell headers minus the HEC byte. The VPI/VCI is actually stored as a single 28 bit field and is only shown as individual fields in the table above for clarity regarding how the actual data is stored in memory.

(12 bits) VPI (4 high order bits of GFC if UNI)	(16 bits) VCI	(4 bits) PTI/CLP
int8u txType	(24 bits) Unused	

(14 words) Unused

Figure 5. Merge Space Usage

The merge space is used to communicate two things, the egress cell header and the type of cell to be transmitted. There are only two cell types recognized by this application, User Data and OAM/RM. The cell type is conveyed by the txType field. The VPI/VCI is actually stored as a single 28 bit field and is only shown as individual fields in the table above for clarity regarding how the actual data is stored in memory.

Control space, as mentioned above is present in varying amounts for each of the SDP components. Not all of them are used. In this application, only the control spaces for RxSync and RxByte are used to communicate initialization information to the SDP. The second byte of control space for RxBit is used as a means of communicating to the XPRC that the CPRC has finished initialization. The first two bytes of RxSync control space are used by the CPRC to provide the delta and alpha count values which drive the cell delineation processing. The third byte holds the current state of the cell delineation state machine. The first byte of RxByte control space holds the CPRC Id and table Id to be used in launching VC lookups. The CPRC Id is in the high order nibble and the table id is in the low order nibble. All other control space is unused.

3.2 XPRC Processing

The XPRC processing performs the chip configuration and initialization chip and then enters an endless loop for ongoing runtime support. The initialization begins by initializing only kernel services. Initialization then begins waiting for the host processor to indicate that it has completed configuration of the Q-5. Currently, this function is implemented by the commSvc applications component and relies on key variables being the first declarations on the stack for the runtime XPRC main program. This approach is expected to change in future releases when the new C-Ware Development System (CDS) driver is integrated with the Traffic Management Application Programming Interface (API).

Table, buffer, fabric, and traffic management services are then initialized. Processing then looks for addition arguments from the host, or the command line in simulation, to establish the fabric id for this processor. If the additional arguments are not provided, the fabric will not be enabled and the application will run with only the front ports. The serial bus is then configured for connection to the Field Programmable Gate Array (FPGA) on the Physical Interface Module (PIM) and the PIM is then taken out of loopback. The TLU VC Table is then loaded with the routes from the offline table building and the queues for the QMU and buffer pools for the BMU are configured and allocated. For full details on queue and buffer pool allocations and use please see the sections of the document for the QMU and BMU. For full details on the tables, their keys, and their contents please see the section of the document for the TLU. The fabric port is then configured for back-to-back operation.

Individual channel processor configuration then begins. From the point of view of port configuration there is little difference between an OC-12 port and an OC-3 port. Each channel processor is configured for SONET support and scrambling support. The transmit large First-In-First-Out (FIFO) watermark is established and auto-insertion of idle cells is enabled in the TxSONET block for when there is no traffic to place in the SONET frame. In addition, for OC-12, automatic token passing is enabled for the transmit SDP. The control spaces for the RxSync and RxByte components are also initialized at this point.

As final initialization steps, the channel processors are loaded with their code, SONET monitor processing on the XPRC is initialized, the CPRCs are started, and the XPRC begins waiting for each CPRC to complete initialization. For full details on SONET monitor processing, please see the applications component documentation for the SONET monitor component. Once a CPRC completes its initialization the XPRC enables that CPRC's SDP components. Once all CPRC's have completed initialization, the XPRC enables the fabric and begins the runtime support section of its processing.

Kernel services is set up to support a timer event in support of SONET monitor processing and processing enters an endless loop for ongoing runtime support.

The runtime support for this application consists of five functions, print service to print text I/O messages from the channel processors, processing of SONET monitor messages from the channel processors, soaking on and off of SONET monitor alarms, dequeuing and discarding of traffic discarded by the Q-5's active queue management, and dequeuing and discarding of OAM/RM traffic.

3.3 C-5e Configuration

The following sections provide details on configuration and use of the non-programmable components of the C-5e.

3.3.1 QMU

NOTE: Descriptors for this application are currently configured to be 32 bytes due to a limitation on the part of the Q-5 FPGA. The descriptors only actually use 16 bytes and can be reduced to that size once the Q-5 supports it. The descriptor size is defined in appConfig.h and the descriptor record structure, with the extra unused fields, is defined in atmIf.h.

QMU configuration for this application must be kept in careful alignment with the Q-5 configuration for the application. Each queue allocated and configured on the QMU essentially equates to a VOP configured on the Q-5. The first 8 CPRCs are unused and have no queues allocated to them. The OC-12 port is served by the next 4 CPRCs and the 4 OC-3 ports are served by the last 4 CPRCs. The OC-12 port is allocated a single queue. Each OC-3 port is allocated a single queue. The fabric port is allocated 7 queues. The XPRC is allocated three queues.

3.3.1.1 OC-12/OC-3/FP Queue Usage

The OC-12 port, each OC-3 port, and the FP port only use the queues to receive traffic for transmission by the transmit processing. All descriptors dequeued from these queues are assumed to use the Cell Descriptor format which is depicted in Figure 6.

BsBufHandle Bh		
Int16u pad1	int8u payloadType	int8u crc10Err
CellHeader CellHdr		
int32u VcIndex		
Unused		
Unused		
Unused		
Unused		

Figure 6. IP Domain Descriptor Structure

The usage of some of the fields is dependent on the forwarding destination. The bufHandle always provides a handle to the buffer containing the packet to be transmitted. The payloadType field defines whether the transmit processing is to use the switched cell processing or the OAM/RM cell processing in the TxByte component. The VcIndex field is only used when the cell is destined for transmission over the fabric and provides the queue id to be used for enqueuing when received by the destination fabric. The CellHdr field holds the egress cell header from the table route for cells to be transmitted. When the cell is an OAM/RM cell being forwarded to the XPRC, the CellHdr field holds the ingress cell header. The crc10Err field only has meaning when the cell is an OAM/RM cell being forwarded to the XPRC and contains the result of the CRC 10 check performed by the SDP.

3.3.1.2 XPRC Queue Usage

The XPRC has three queues, one for SONET monitor traffic, one for discard processing, and one for OAM/RM cells. Please see the SONET monitor documentation for details regarding it's queue usage. The discard processing is only concerned with getting a handle to the buffer so that the buffer can be freed. Currently, the OAM/RM cell processing is identical to the discard processing and simply discards the associated buffer. If OAM/RM processing were to be implemented on the host, the XPRC could then forward these cells to the host for further processing.

3.3.2 BMU

The initialization phase of the XPRC processing allocates a buffer pool for use by each CPRC, including those not currently activated by this application. This makes it simple for the CPRC processing to determine the pool to be used since the pool Id will match the CPRC Id. The CPRC then initializes the pool for it's use as part of the channel processor initialization processing.

The XPRC does not currently allocate any buffers as part of its processing and therefore does not allocate any buffer pools for it's own use. Four buffer pools are allocated for use by the FP.

The number of buffers needed in each pool and the buffer size are things which can be very application dependent. The number of buffers and the buffer size are defined in appConfig.h to make easy to modify when converting the reference application to a specific customer application.

The buffer pools are currently configured the same for both the OC-12 and OC-3 ports. Each pool is allocated a buffer pool with 128 buffers where each buffer is 64 bytes in size. For the FP, each pool is allocated with 256 buffers of size 64 bytes.

3.3.3 TLU

The current application has only one tables, the VC lookup table. The table is created by the offline processing and restored by the XPRC initialization processing. The offline table building is coded such that the table id/algorithm id used for the VC lookup table is always 0.

The following section details the key and entry contents for the VC lookup table and the request/response slot usage for the channel processors for launching TLU commands and receiving TLU responses.

3.3.3.1 VC Lookup Table

The VC lookup table provides exact match behavior on the VC ID which is formed from the VPI/VCI from the cell header and the ingress port id. The key field for the VC table is 32 bits. The detail composition of the key is detailed below in Figure 7.

(12 bits) VPI	(16 bits) VCI	(4 bits) Port Id

Figure 7. VC Id Composition

The routes in this table supply TQ ids that can enqueue to either the OC-12, OC-3, or FP transmit processing. The table entry structure always provides a flags field, a VC Index, a TQ id, and an egress cell header. The flags field provides for up to 8 bit flags. Currently only two are used, one to allow disabling of the route without deleting it from the table and a second to indicate that the route is to be directed out the fabric port. When the flags indicate that the route is for traffic going out the fabric port, the table entry also provides a fabric queue id to be used for enqueuing to the fabric. The table also contains a number of fields which are not used by the processing but aid in making the table readable. Specifically, the table contains a field identifying the VC as CBR, VBR-Real-Time (RT), VBR-Non-Real-Time (NRT), or Unspecified Bit Rate (UBR), and individual fields for the egress VPI/VCI. The structure of the table entry is detailed in Figure 8 below where each row represents 32 bits.

int8u Flags	int8u TrafficClass	int16u VcIndex	
Int32u QueueId			
Int16u Vpi		int16u Vci	
Int32u atmEgressKey			
int32u fabricQueueId			
int32u pad1			

Figure 8. VC Table Entry Structure

The queueId field provides the TQ id to be used for enqueuing to the Q-5 for local routes and the TQ id to be used for enqueuing by the receiving fabric for remote routes. The VCIndex field provides an integer unique identifier for each VC in the table. The atmEgressKey provides the egress VPI and VCI already formatted as a cell header whereas the Vpi and Vci fields provide the egress VPI/VCI as individual fields.

3.3.3.2 Ring Bus Slot Usage

Each channel processor has four request slots and eight response slots available for sending commands to the TLU and receiving responses. Each slot is supports data of 8 bytes in size. The SDP only has access to the first two request slots and none of the response slots (which means all TLU responses must be processed by channel processors).

The slot usage for launching lookups in the SDP and receiving the results of those lookups at the channel processor is the same for both the OC-3 and OC-12 ports. The slot usage for all VC lookups are shown below in Figure 9.

Lookup	Request Slot(s)	Response Slot(s)
VC Lookup	0	0

Figure 9. Ring Bus Slot Usage

3.4 Host Processing

Host processing for this application is only concerned with initialization phase configuration of the Q-5. The configuration of the Q-5 can be broken up into three categories, configuration for scheduling of traffic destined for the OC-12 port, configuration for scheduling of traffic destined for the OC-3 ports, and configuration for passing traffic to the XPRC in support of the SONET monitor and OAM/RM processing. The configuration for traffic headed to the FP is a mirror of the combination of the OC-12 and OC-3 configurations (i.e., three FP VOPs mirror the three OC-12 scheduling trees and 4 FP VOPs mirror the four OC-3 scheduling trees). The configuration for each category is discussed further in the following sections.

There is currently no support in the application for dynamic reconfiguration of the scheduling. All Q-5 configuration happens at initialization time prior to allowing the XPRC runtime processing to complete it's initialization processing.

3.4.1 OC-12 Scheduling

Figure 10 below shows the scheduling trees used to schedule the traffic destined for transmission by the OC-12 port.

Half of the customers on an L1 are configured with CBR/UBR and the other half are configured with VBR/UBR. This is done as every other customer. There is no hardware driven reason for the ratio nor the order in which they are alternated. Since it is the policing which determines the nature of the VC in this case, and each TQ can be individually policed, there is nothing which prevents a CBR/VBR configuration either although that is not done in the reference application.

The rates used to police the CBR and VBR VCs and the rates used to shape traffic for each customer vary across the L1 scheduler but are identical for each of the three L1s. This is also not required but is done to simplify the programming for the reference application. In general, the rates decrease moving from the leftmost leg of the L1 (leg 0) to the rightmost leg (leg 1023).

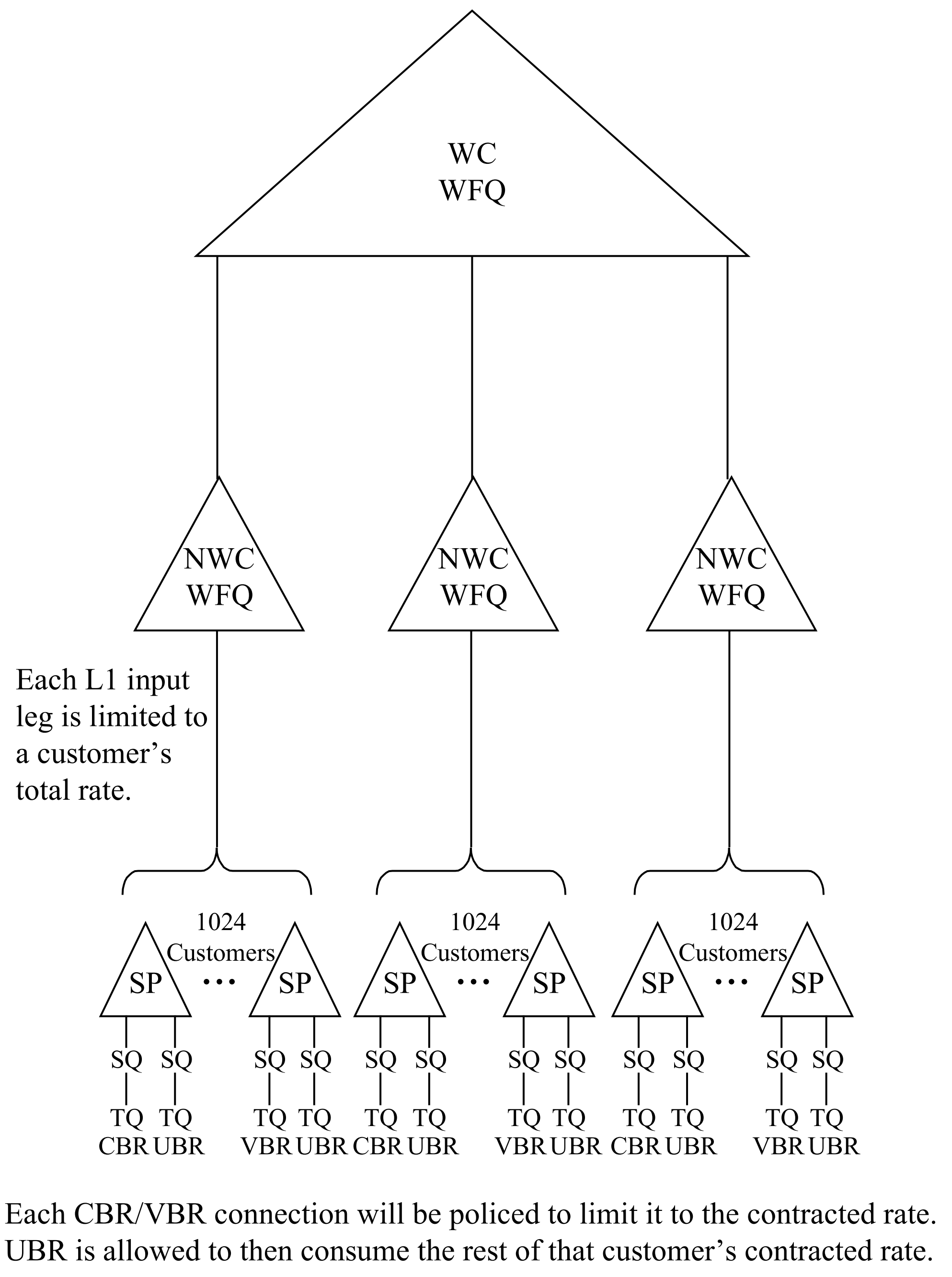
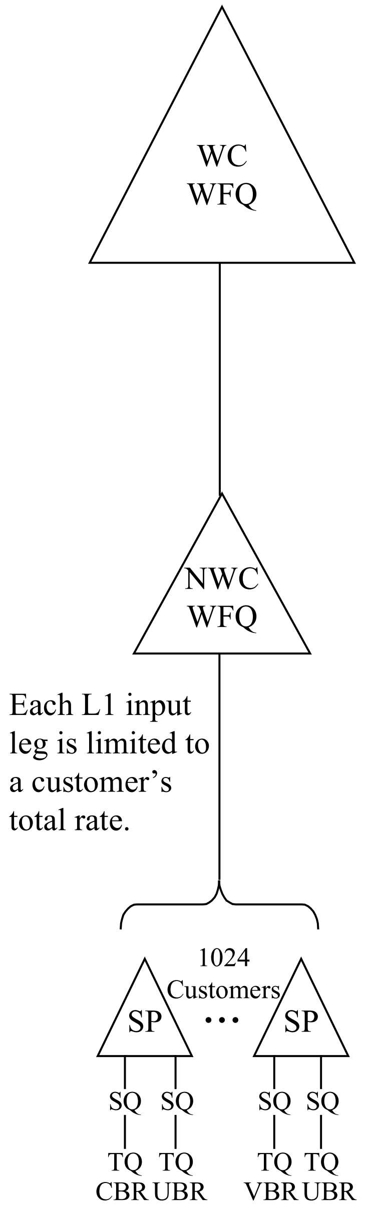

WC
WFQ
NWC
WFQ
NWC
WFQ
NWC
WFQ
Each L1 input leg is limited to a customer's total rate.
1024 Customers
1024 Customers
1024 Customers
SP
SP
SP
SP
SP
SP
SQ SQ SQ SQ SQ SQ SQ SQ SQ SQ SQ SQ
TQ TQ TQ TQ TQ TQ TQ TQ TQ TQ TQ TQ
CBR UBR VBR UBR CBR UBR VBR UBR CBR UBR VBR UBR
Each CBR/VBR connection will be policed to limit it to the contracted rate.
UBR is allowed to then consume the rest of that customer's contracted rate.

Figure 10. OC-12 Port Scheduling

The OC-12 customer rates, in bytes per second, used for shaping are 10 MB, 1 MB, 100 KB, 64 KB, and 56 KB. The rates, in bytes per second, used for policing CBR traffic for OC-12 are 10 MB, 5 MB, 1 MB, 256 KB, 128 KB, 100 KB, 75 KB, 64 KB, 56 KB, 32 KB, and 16 KB. The rates, in bytes per second, used for policing VBR traffic for OC-12 are (PCR/SCR) 10 MB/1 MB, 1 MB/256 KB, 100 KB/50 KB, 100 KB/10 KB, 64 KB/16 KB, 64 KB/8 KB, 32 KB/4 KB, 16 KB/8 KB, and 16 KB/4 KB. Most details for the scheduling trees can be found in tmcParams.h with the rest of the details being in tmcConfig.h and tmcConfig.c. Specifically, the customer rates for shaping are controlled by the oc12L1SchedulerParams structure and the policing rates are controlled by the oc12Customers structure, where both structures are in tmcParams.h

3.4.2 OC-3 Scheduling

Figure 11 below shows the scheduling tree used to schedule the traffic destined for transmission by an OC-3 port. Each port uses one of these trees to independently schedule that port. The OC-3 port scheduling follows the same conventions mentioned for OC-12, every other customer is either CBR/UBR or VBR/UBR and rates decrease from left to right on the L1 scheduler.



Each CBR/VBR connection will be policed to limit it to the contracted rate. UBR is allowed to then consume the rest of that customer's contracted rate.

Figure 11. OC-3 Port Scheduling

The OC-3 customer rates, in bytes per second, used for shaping are 100 KB, 64 KB, and 56 KB. The rates, in bytes per second, used for policing CBR traffic for OC-3 are 100 KB, 75 KB, 64 KB, 56 KB, 32 KB, 16 KB, and 16 KBits. The rates, in bytes per second, used for policing VBR traffic for OC-12 are (PCR/SCR) 100 KB/50 KB, 100 KB/10 KB, 64 KB/16 KB, 64 KB/8 KB, 32 KB/4 KB, 16 KB/8 KB, and 16 KB/4 KB. Most details for the scheduling trees can be found in tmcParams.h with the rest of the details being in tmcConfig.h and tmcConfig.c. Specifically, the customer rates for shaping are controlled by the oc3L1SchedulerParams structure and the policing rates are controlled by the oc3Customers structure, where both structures are in tmcParams.h.

3.4.3 XPRC Scheduling

It is somewhat of a misnomer to refer to the following as scheduling. It is in fact simply a pass through with one TQ contributing solely to the output of one VOP with no shaping or policing occurring. The same scheduling approach is used by both SONET Monitoring and OAM/RM cell forwarding. The SONET queue is used by SONET monitoring processing on the CPRCs to communicate SONET defect information to the SONET monitoring processing on the XPRC. The OAM/RM queue is used by the CPRCs to forward all received OAM/RM cells to the XPRC. The scheduling tree is depicted below in Figure 12.

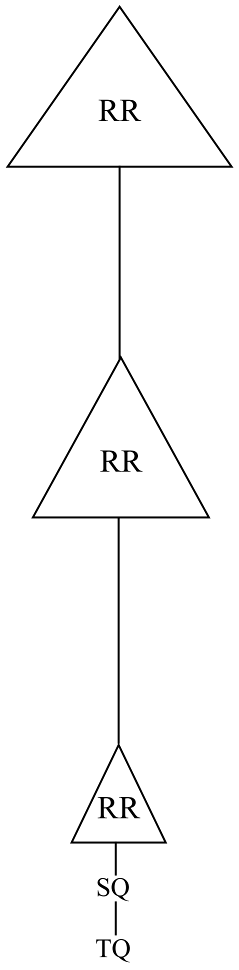


Figure 12.   XPRC Queue Scheduling

4 Other Information

4.1 Offline Tables

In order to support testing in a simulation environment, and to ease testing on hardware with test equipment, this application is built with a number of static routing addresses already entered into it's routing table. This is accomplished by using tables constructed offline by code which interacts with a TLU model to generate the raw data which would represent these routes in the TLU memory. This raw data is then compiled into the application as data and written to the TLU as part of application initialization.

To simplify testing, VPI/VCI values have been assigned in blocks for a given route and type (i.e., CBR VCs into the OC-12 port and out the first OC-3 port will all have the same VPI and incrementing VCI values. Defines have been created in appConfig.h marking rate change boundaries in the TQ number space to aid in creating routes for the different rate classes. The table below summarizes the routes in the offline table.

Description	Ingress VPI	Ingress VCI	Egress VPI	Egress VCI
VBR VCs In OC-12 Port Out OC-3 Port 0	32	32 – 47	64	64 – 79
CBR VCs In OC-12 Port Out OC-3 Port 0	64	64 – 79	96	96 - 111
UBR VCs For VBR Customers In OC-12 Port Out OC-3 Port 0	80	32 – 47	112	64 – 79
UBR VCs For CBR Customers In OC-12 Port Out OC-3 Port 0	80	64 – 79	112	96 – 111
VBR VCs In OC-12 Port Out OC-3 Port 1	33	32 – 47	65	64 – 79
CBR VCs In OC-12 Port Out OC-3 Port 1	65	64 – 79	97	96 - 111
UBR VCs For VBR Customers In OC-12 Port Out OC-3 Port 1	81	32 – 47	113	64 – 79
UBR VCs For CBR Customers In OC-12 Port Out OC-3 Port 1	81	64 – 79	113	96 – 111
VBR VCs In OC-12 Port Out OC-3 Port 2	34	32 – 47	66	64 – 79
CBR VCs In OC-12 Port Out OC-3 Port 2	66	64 – 79	98	96 - 111
UBR VCs For VBR Customers In OC-12 Port Out OC-3 Port 2	82	32 – 47	114	64 – 79
UBR VCs For CBR Customers In OC-12 Port Out OC-3 Port 2	82	64 – 79	114	96 – 111
VBR VCs In OC-12 Port Out OC-3 Port 3	35	32 – 47	67	64 – 79
CBR VCs In OC-12 Port Out OC-3 Port 3	67	64 – 79	99	96 - 111

Description	**Ingress VPI**	**Ingress VCI**	**Egress VPI**	**Egress VCI**
UBR VCs For VBR Customers In OC-12 Port Out OC-3 Port 3	83	32 – 47	115	64 – 79
UBR VCs For CBR Customers In OC-12 Port Out OC-3 Port 3	83	64 – 79	115	96 – 111
VBR VCs in OC-3 Port 0 Out OC-12 Port	32	32 – 41	64	64 – 73
CBR VCs in OC-3 Port 0 Out OC-12 Port	64	64 – 78	96	96 – 110
UBR VCs For VBR Customers In OC-3 Port 0 Out OC-12 Port	80	32 – 41	112	64 – 73
UBR VCs for CBR Customers In OC-3 Port 0 Out OC-12 Port	80	64 – 78	112	96 – 110
VBR VCs in OC-3 Port 1 Out OC-12 Port	33	32 – 41	65	64 – 73
CBR VCs in OC-3 Port 1 Out OC-12 Port	65	64 – 78	97	96 – 110
UBR VCs For VBR Customers In OC-3 Port 1 Out OC-12 Port	81	32 – 41	113	64 – 73
UBR VCs for CBR Customers In OC-3 Port 1 Out OC-12 Port	81	64 – 78	113	96 – 110
VBR VCs in OC-3 Port 2 Out OC-12 Port	34	32 – 41	66	64 – 73
CBR VCs in OC-3 Port 2 Out OC-12 Port	66	64 – 78	98	96 – 110
UBR VCs For VBR Customers In OC-3 Port 2 Out OC-12 Port	82	32 – 41	114	64 – 73
UBR VCs for CBR Customers In OC-3 Port 2 Out OC-12 Port	82	64 – 78	114	96 – 110
VBR VCs in OC-3 Port 3 Out OC-12 Port	35	34 – 41	67	66 – 73
CBR VCs in OC-3 Port 3 Out OC-12 Port	67	66 – 78	99	98 – 110
UBR VCs For VBR Customers In OC-3 Port 3 Out OC-12 Port	83	34 – 41	115	66 – 73
UBR VCs for CBR Customers In OC-3 Port 3 Out OC-12 Port	83	66 – 78	115	98 – 110
CBR VCs In OC-12 Port Out Fabric Port	96	96 – 103	160	160 - 167

Figure 13. VC Lookup Table

It should be noted that these routes were not selected to be representative of real world routing tables but simply to ease testing of the application. Routes are not provided for all potential traffic paths for this application. For instance, there are no routes which go in one OC-3 port and out another OC-3 port although that is a legitimate path for traffic in this application. This is mostly due to the fact that the offline table is compiled into the application as a data structure which limits how large it can be. It should also be noted that since the application runs only in simulation currently, prohibiting long, lengthy testing and making complicated traffic testing difficult, not all routes in the table have been tested. In particular, the fabric routes have not been exercised.

4.2 Issues, Constraints, And Future Enhancements

4.2.1 Static Scheduling Hierarchy

The current implementation establishes the complete scheduling hierarchy at initialization and no changes are supported during runtime. Eventually, the Q-5 will support dynamically re-configuration and the Host portion of this application should be enhanced to support a user interface allowing re-configuration commands to be issued.

4.2.2 Static Routing Table

Similar to the scheduling hierarchy, the routes in the VC table are currently created offline and the table is static for the duration of the run. As noted in the TLU section, this also limits the number of routes in the table due to DMEM constraints. The host portion of this application should be enhanced to populate the table rather than using offline tables and should populate the table with routes that would allow accessing of all paths through the scheduling trees and all valid data paths through the application.

4.3 Application Files And Binaries

The following section described the files provided as part of this reference application and some of the binaries produced as a result of building it. For details on how to build the application, please see the README file and the Build System Conventions document.

Filename	Short Description
tables.h	Prototypes for functions in tables.c
tables.c	Data and routines for creating and entering routes in offline tables
offline.c	Main program for offline table building
Makefile	Makefile for building offline tables

Figure 14. Offline Tables Files

The above files are located in apps/atmCellSwitchQ/offline.

Filename	Short Description
Tlu.State	Offline table artifact for simulation
config	Simulation configuration file

atmCellSwitchQQ.dsc	Package description file for packaging binaries
Makefile	Makefile for building application binaries
sim.in	Input command file for simulation runs
acceptAtmCellSwitchQ.expected	Expected file for application's automated accept test
run.sh	Script file for automating accept test

Figure 15. Run Files

The above files are located in apps/atmCellSwitchQ/run.

Filename	**Short Description**
Makefile	Makefile for building input patterns for simulation

Figure 16. inPatterns Files

The above file is located in apps/atmCellSwitchQ/run/inPatterns.

Filename	**Short Description**
rcSdpAtmApiIf.h	Data structures for merge space, extract space, and control space usage by the application
atmVcTable.h	Data structures for keys and data entries for the VC Lookup Table
atm.h	Data structures for cell header and statistics
atmIf.h	Data structure for descriptors passed through the queues in this application
appConfig.h	Defines used to customize the application
tmHostTestCommDefs.h	Defines used for current approach for XP and Host communications (expected to change in the future).

Figure 17. Common Header Files

The above files are located in apps/atmCellSwitchQ/chip/np/inc.

Filename	**Short Description**
tle_writes.h	Offline table artifact for restoring tables
tle_restore.h	Offline table artifact for restoring tables
xpMain.c	Source code for all XPRC processing
Makefile.in	Makefile support file for building XPRC binaries

Figure 18. XPRC Files

The above header files are located in apps/atmCellSwitchQ/chip/np/xprc/inc. The above source files are located in apps/atmCellSwitchQ/chip/np/xprc/src.

Filename	Short Description
atmTxByte.c	Microcode for transmitting switched cells and OAM/RM cells
atmRxSync.c	Microcode for cell delineation state machine for extracting cells from the data stream
atmRxByte.c	Microcode for receiving cells
Makefile.in	Makefile support file for building SDP binaries

Figure 19. SDP Files

The above source files are located in apps/atmCellSwitchQ/chip/np/sdp/src.

Filename	Short Description
atmCp.h	Prototypes for ATM source code routines
atmOc3CpMain.c	CPRC main program for OC-3 Port CPRCs
atmOc3RxCp.c	Receive processing for OC-3 Port CPRCs
atmOc3TxCp.c	Transmit Processing for OC-3 Port CPRCs
atmOc12CpMain.c	CPRC main program for OC-12 Port CPRCs
atmOc12RxCp.c	Receive processing for OC-12 Port CPRCs
atmOc12TxCp.c	Transmit Processing for OC-12 Port CPRCs
atmInitCp.c	Common ATM Initialization processing
Makefile.in	Makefile support file for building CPRC binaries

Figure 20. CPRC Files

The above header files are located in apps/atmCellSwitchQ/chip/np/cprc/inc. The above source files are located in apps/atmCellSwitchQ/chip/np/cprc/src.

Filename	Short Description
tmcParams.h	Defines and pre-initialized structures for configuring the Q-5 for this application
tmcConfig.h	Variables and defines for holding ids of objects created in configuring the Q-5
tmHostMain.cpp	Main program for host processing to configure the Q-5
tmcConfig.c	Processing which configures the Q-5 for this application

Makefile.in	Makefile support file for building Host binaries

Figure 21. Host Files

The above header files are located in apps/atmCellSwitchQ/host/np/inc. The above source files are located in apps/atmCellSwitchQ/host/np/src.

Filename	**Short Description**
atmCellSwitchQ.pkg	The final packaged image for the C-5e load
atmCellSwitchQQXp.dcp	The binary for the XPRC
atmCellSwitchQOc12Cp.dcp	The OC-12 Port binary for the CPRC
atmCellSwitchQOc3Cp.dcp	The OC-3 Port binary for the CPRC
oc12Ucode.sdp	The OC-12 Port binary for the SDP
oc3Ucode.sdp	The OC-3 Port binary for the SDP
fpCportFdpQ5.sdp	The binary for the FP
tmHost	The Host binary for the application

Figure 22. Binary Files

The above files are created as a result of building the image. The XPRC, CPRC, and SDP binaries for the C-5 are all found in the same variant directory under apps/atmCellSwitchQ/run/bin and the host binary is found in a separate variant directory under the same place. The FP binary is found in the variant directory under apps/components/fabrics/bin. For the conventions determining the name(s) of the variant directories based on the build environment at the time of the build, please see the Build Systems Conventions document.

Appendix A
Acronyms

API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BMU	Buffer Management Unit
CBR	Constant Bit Rate
CDS	C-Ware Development System
CLP	Cell Loss Priority
CPI	C-Ware Programming Interface
CPRC	Channel Processor RISC Core
CRC	Cyclic Redundancy Check
CST	C-Ware Software Toolset
DMA	Direct Memory Access
FIFO	First-In First-Out
FP	Fabric Processor
FPGA	Field Programmable Gate Array
GFC	Generic Flow Control
HEC	Header Error Control
KB	KiloByte
MB	MegaByte
NRT	Non-Real-Time
NWC WFQ	Non-Work Conserving Weighted Fair Queuing
OAM	Operations And Maintenance
PCR	Peak Cell Rate
PIM	Physical Interface Module
PTI	Payload Type Indication
QMU	Queue Management Unit
RISC	Reduced Instruction Set Computer
RM	Resource Management
RR	Round Robin
RT	Real-Time
SCR	Sustainable Cell Rate
SDH	Synchronous Digital Hierarchy
SDP	Serial Data Processor
SONET	Synchronous Optical Network
SP	Strict Priority
SQ	Scheduler Queue
TLU	Table Lookup Unit
TQ	Traffic Queue
UBR	Unspecified Bit Rate
UNI	User Network Interface

VBR	Variable Bit Rate
VC	Virtual Channel
VCI	Virtual Channel Identifier
VOP	Virtual Output Port
VPI	Virtual Path Identifier
WC WFQ	Work Conserving Weighted Fair Queuing
XPRC	eXecutive Processor RISC Core