PM7325

ATLAS-3200

2488 MBIT/S SATURN USER NETWORK INTERFACE ATM LAYER SOLUTION

PROGRAMMER'S GUIDE

CONFIDENTIAL

PRELIMINARY

ISSUE 1: DEC 2000

PUBLIC REVISION HISTORY

Issue No.	Issue Date	Details of Change
1	Dec 2000	Document created.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 REFERENCES

1. PMC-1990553, PMC-Sierra Inc., "OC-48 Saturn User Network Interface ATM Layer Solution Standard Product Data Sheet", Oct 2000, Issue 3

2. ITU-T Recommendation I.610 (02/1999), B-ISDN operation and maintenance principles and functions

3. ITU-T Recommendation I.630 (02/1999), ATM protection switching

4. ATM Forum TM4.1, ATM Forum Traffic Management Specification Version 4.1, 1999.

5. ITU-T Recommendation I.356 (03/00), B-ISDN ATM layer cell transfer performance

6. ITU-T Recommendation I.371 - "Traffic Control and Congestion Control in B-ISDN", May, 1996

7. ATM Forum AF-TM-0121.000 - Traffic Management Specification Version 4.1, March 1999

8. ITU-T Recommendation I.610 - "B-ISDN Operation and Maintenance Principles and Functions", February 1999.

9. Bell Communications Research - Generic Requirements for Operations of Broadband Switching Systems, GR-1248-CORE, Issue 3, August 1996.

10. ATM Forum AF-PHY-0136.000 - UTOPIA 3 Physical Layer Interface, November 1999.

11. PMC-1980495, "POS-PHY Level 3: SATURN Compatible Interface for Packet Over SONET Physical Layer and Link Layer Devices", Issue 4, June 7, 2000.

12. ATM Forum AF-PHY-0039.000 - UTOPIA, An ATM-PHY Interface Specification, Level 2, Version 1.0, June 1995.

13. PMC-880901 "Telecom System Block (TSB) User Manual", Issue 6, March 28, 1997.

2 GLOSSARY

AIS	Alarm Indication Signal. An OAM cell that indicates to downstream entities that there is a fault in the VC.
CC	Continuity Check. An OAM cell sent through the network so that downstream entities may differentiate between a failure and periods of low user cell traffic.
FM	Fault Management. The mechanism used by the network to inform management entities and other network equipment of faults within the network. Consists of AIS, RDI, CC, and LB cell flows.
GCRA	Generic Cell Rate Algorithm. The algorithm used to police the cell transmission rate. Also known as the Leaky Bucket Algorithm.
GFR	Guaranteed Frame Rate. The algorithm used to police the packet transmission rate.
IBCIF	Input Backwards Cell Interface. A component of the ATLAS-3200 that receives cells from another ATLAS-3200 that is transmitting in the opposite direction.
ICIF	Input Cell Interface. A component of the ATLAS-3200 that receives cells.
IMCIF	Input Microprocessor Cell Interface. A component of the ATLAS-3200 that receives cells or packets from the microprocessor and inserts them into a connection.
OAM	Operations and Maintenance. The maintenance of VCs within the network.
OBCIF	Output Backwards Cell Interface. A component of the ATLAS-3200 that outputs cells to another ATLAS-3200 that is transmitting in the opposite direction.
OCIF	Output Cell Interface. A component of the ATLAS-3200 that outputs cells.
OMCIF	Output Microprocessor Cell Interface. A component of the ATLAS-3200 that receives cells or packets from a connection and sends them to the microprocessor.

PM	Performance Management. The mechanism used by the network to monitor the performance parameters of a particular VC.
RDI	Remote Defect Indication. An OAM cell sent to an upstream entity at the OAM flow endpoint to indicate that there is a fault in the VC.
SDQ	Scaleable Data Queue. A memory bank in the ATLAS-3200 that is configurable into variable sized FIFOs for buffering the input and output interfaces.
VC	Virtual Connection. This refers to either a Virtual Path Connection (VPC) or a Virtual Channel Connection (VCC).
VCC	Virtual Channel Connection. A virtual connection between two network elements. A virtual channel connection is normally a constituent member of a virtual path connection, where the VPC consists of one or more VCCs. This is sometimes known as an F5 connection.
VPC	Virtual Path Connection. A virtual connection between two network elements. A virtual path connection may span one or more physical links. This is sometimes known as an F4 connection.

3 INTRODUCTION

This document is a guide to programming the software for a microprocessor that interfaces to an ATLAS-3200 chip. It is intended to ease the integration of the ATLAS-3200 into system designs and to enable optimum performance to be achieved. Suggested programming algorithms, configuration settings and discussion of the chip's operation are presented.

The document's major sections are an overview of the ATLAS-3200, a description of the microprocessor interface, a high level programming section, and a component interface programming section. The Programming High Level Operations section discusses configuration settings and algorithms related to programming the high level functionality of the ATLAS-3200. The Programming Component Interfaces section provides the algorithms for reading and writing each of the ATLAS-3200 interfaces.

3.1 Scope

The ATLAS-3200 Programmer's Guide supplements the ATLAS-3200 Data Sheet [1] with additional information for software programming.

Typical system applications will use more than one ATLAS-3200, however, due to the variety of possible configurations this document only discusses software programming for a single chip. Programming the routines to coordinate multiple ATLAS-3200s must be done on an application specific basis.

Although every effort has been taken to ensure the consistency between this document and the ATLAS-3200 Data Sheet [1], some discrepancies may occur. In case of inconsistencies between this document and the ATLAS-3200 Data Sheet [1], the information in the Data Sheet [1] should be considered accurate and takes precedence over information provided in this document.

Please contact a PMC-Sierra Applications Engineer for information not covered in this document.

3.2 Target Audience

This document is prepared for software and system designers who are using the ATLAS-3200 chip. Some prior knowledge of the ATM protocol, as well as some knowledge of the C programming language and computer operation is necessary to fully understand this document.

3.3 Numbering Conventions

The following numbering conventions are used throughout this document:

- Decimal 129, 6, 12
- Binary 011, 1011 (when distinguished from decimal by the context)

  or 0b011, 0b1011
- Hexadecimal 0x1FE2

3.4 Pseudo-Code Conventions

The pseudo-code in this document is shown in a C-like syntax. The pseudo-code segments are provided as a reference in order to understand a particular procedure or concept. The pseudo-code, however, is not compile-ready and is not provided for all sections. The pseudo-code segments are contained in the Example Routines sections at the end of the major sections to which they relate.

Function calls that are made to other pseudo-code routines that are defined in this document are highlighted in bold. Function calls that are not highlighted in bold indicate that the routine is not provided. The routines that are not provided are either self explanatory or application specific.

4 ATLAS-3200 OVERVIEW

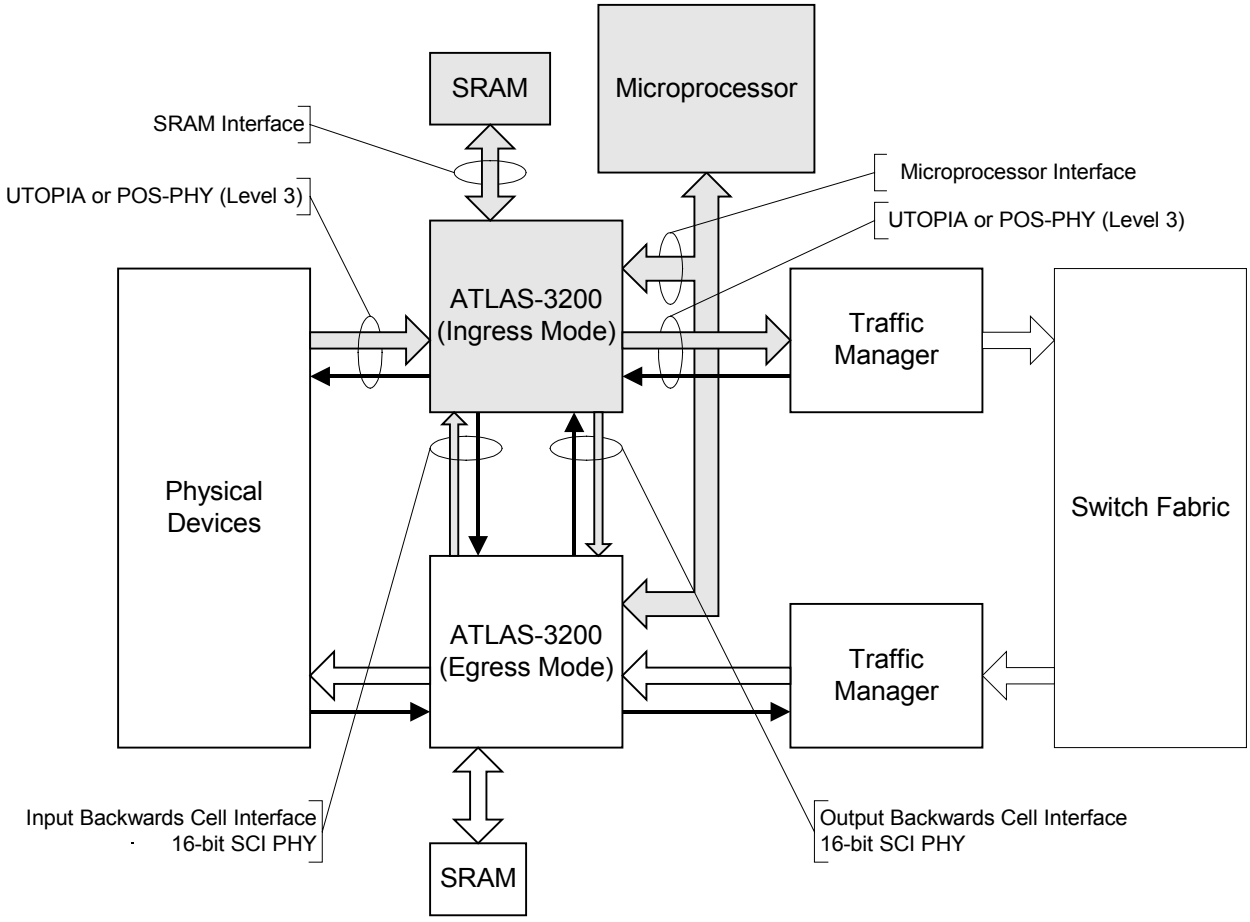
The ATLAS-3200 is an integrated circuit that implements ATM Layer functions that include header translation, cell rate policing, per-connection cell counting and I.610 compliant OAM requirements for up to 64K virtual connections. It is a uni-directional part that is intended to be situated between the physical layer devices and the traffic manager.

The sections below discuss a typical system application, the operating modes, and the data structures. These sections are intended to serve as a reference for general understanding of the ATLAS-3200, and are referred to frequently in the rest of the document.

4.1 System Application Example

A typical system application is shown in Figure 1. The ATLAS-3200s are typically used in pairs. One chip of the pair receives cells/packets from PHY Layer devices and sends them towards the traffic manager/switch core. This chip is said to be in Ingress mode. The other chip of the pair receives cells/packets from the traffic manager/switch core and sends them to the PHY Layer devices. This chip is said to be in Egress mode. The Backwards Cell Interfaces allow the ATLAS-3200s to insert cells into the opposite direction cell flow. Each ATLAS-3200 has an external SRAM bank for holding part of the connection information (the other part being held in the internal embedded DRAM) and a Microprocessor Interface to provide the connection to a microprocessor. This document is a guide to programming the software for this microprocessor.

Figure 1 - ATLAS-3200 System Application



4.2 Operation Modes

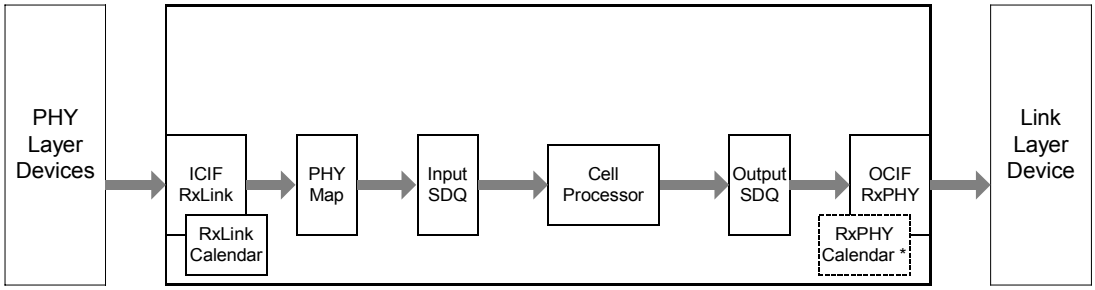
The ATLAS-3200 can be configured in four Operating Modes based on the signaling type and the direction of cell transmission with respect to the switch core. The signaling type can be either UTOPIA or POS-PHY, and the transmission direction can be either Ingress or Egress. UTOPIA Level 3 is used for transferring fixed-length ATM cells; and POS-PHY Level 3 is used for transferring variable-length packets. Applications that require the ATLAS-3200 to transfer mixed cell and packet traffic should use POS-PHY signalling.

In each operating mode different components and data paths within the ATLAS-3200 are used. A diagram summarizing each operating mode is provided in Figure 2.

All the programming routines in this document for a single ATLAS-3200, however, they apply equally to all modes of operation (Ingress or Egress and UTOPIA or POS-PHY) unless noted otherwise.

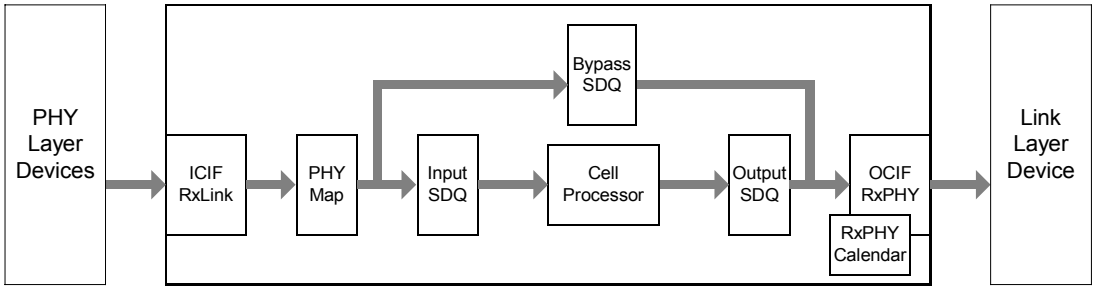
Figure 2 - Operating Mode Components and Data Paths

Ingress UTOPIA mode

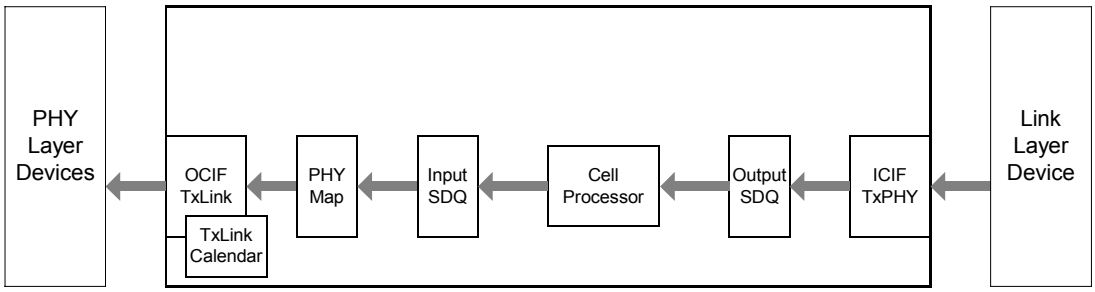


* Optional calendar. Only used if RxPHY slave is configured as a single PHY.

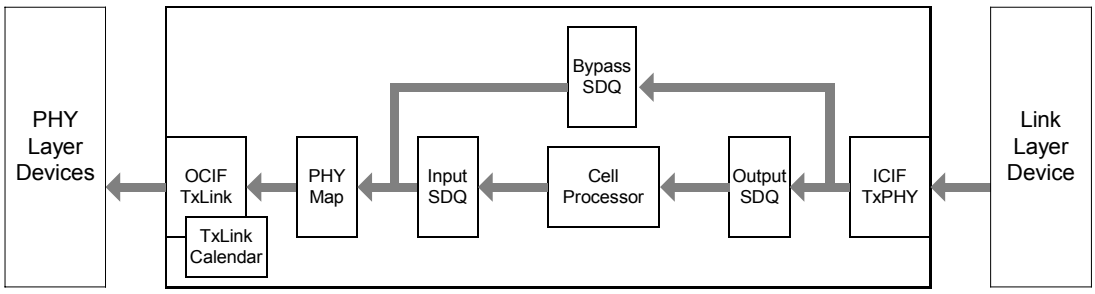
Ingress POS-PHY mode



Egress UTOPIA mode



Egress POS-PHY mode



4.3 Data Structures

The ATLAS-3200's operation relies on data structures located in internal and external memory. One bank of external SRAM stores the VC Linkage Table and Search Tables. All other data structures are stored in memory internal to the ATLAS-3200.

The microprocessor can access all data structures indirectly through these registers. The indirect access interface to each data structure is described in Section 7 Programming Component Interfaces. The figures in this section show all the data structures that are accessible by the microprocessor.

Figure 3 shows the memory locations and relationships between the data structures directly related to the Virtual Connection Record Tables. The dashed lines in Figure 3 indicate fields in the data structures that point to other locations in memory. One example of each pointer type is shown. The VC Linkage Table should be viewed conceptually as an extension of the VC Table, but resides in external SRAM instead of the internal DRAM for performance reasons. Thus, the information in location 0x123 (for example) of the linkage table is simply an extension of the information in location 0x123 of the VC Table.

The FIFOs contained in the ATLAS-3200 are shown in Figure 4, and all the remaining internal data structures are shown in Figure 5.

A description of the external memory sizing is shown in Figure 6.

Figure 3 - VC Related Data Structures

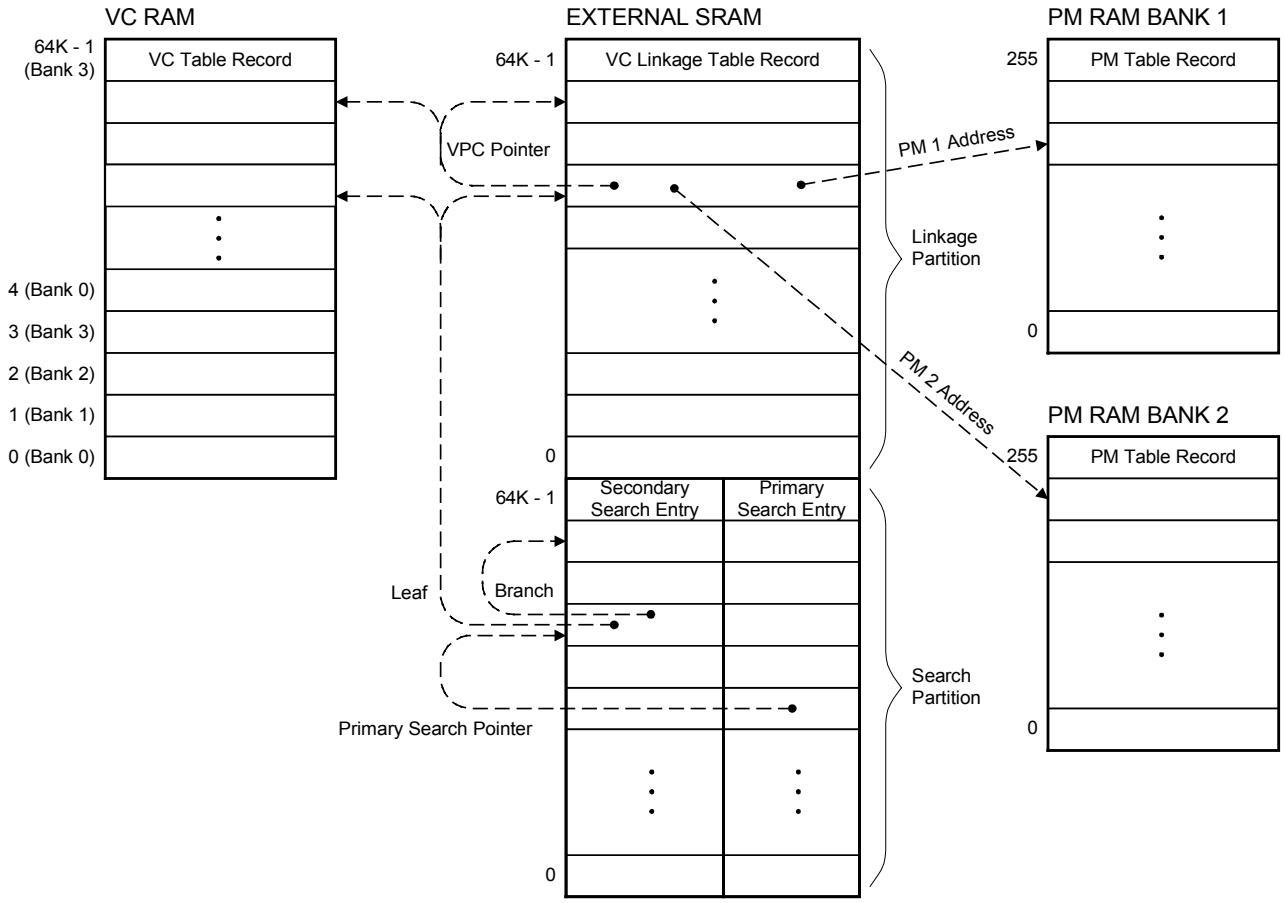


Figure 4 - FIFO Data Structures

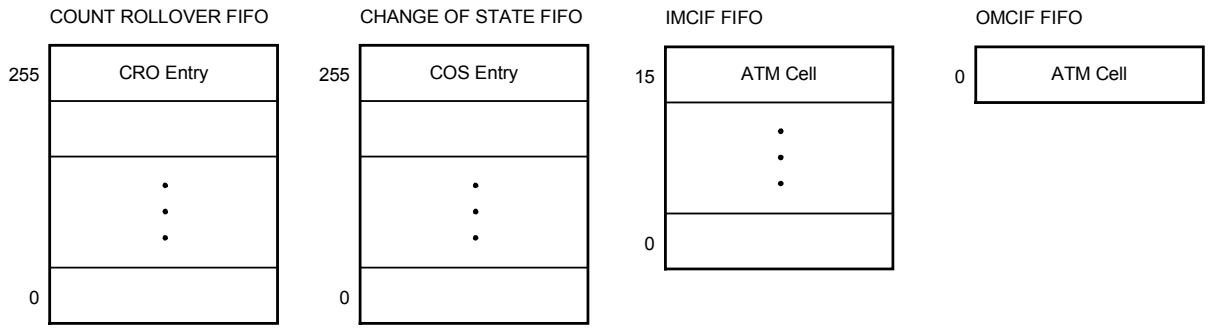
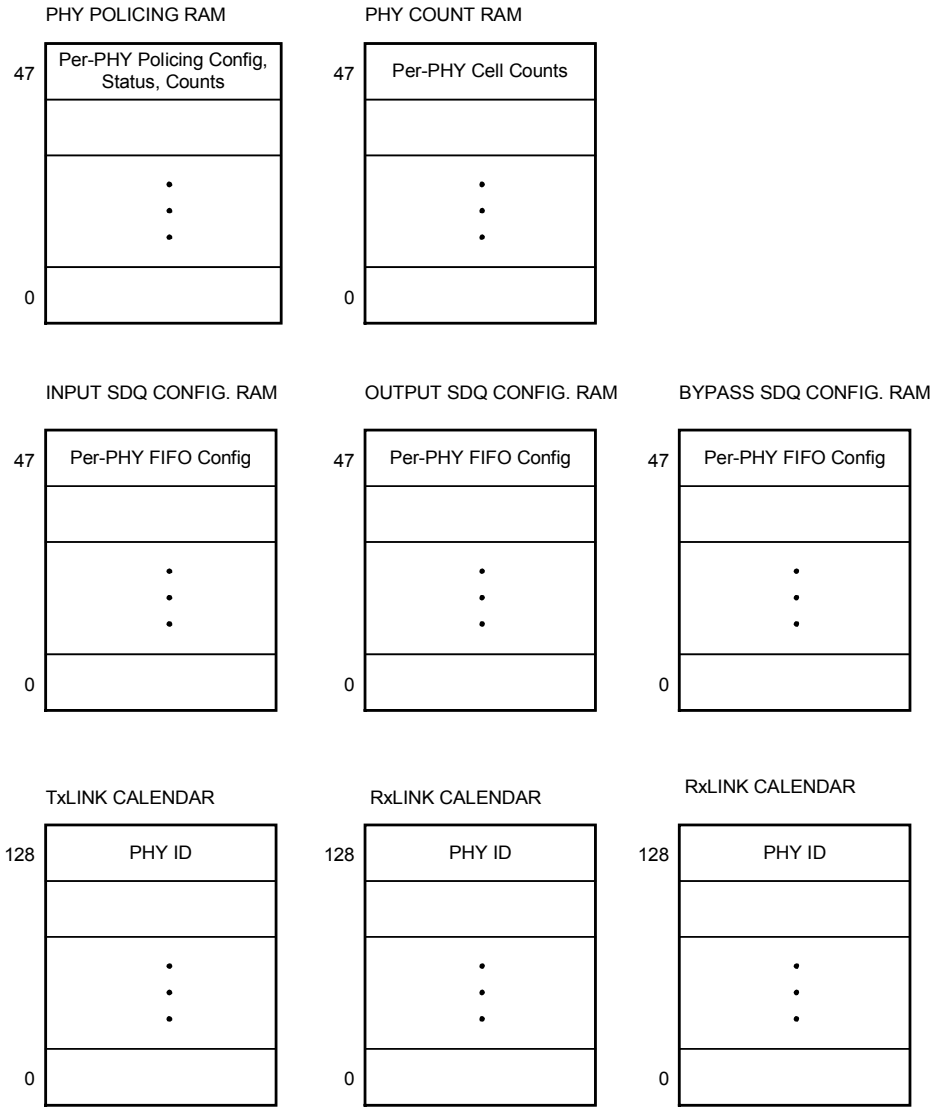


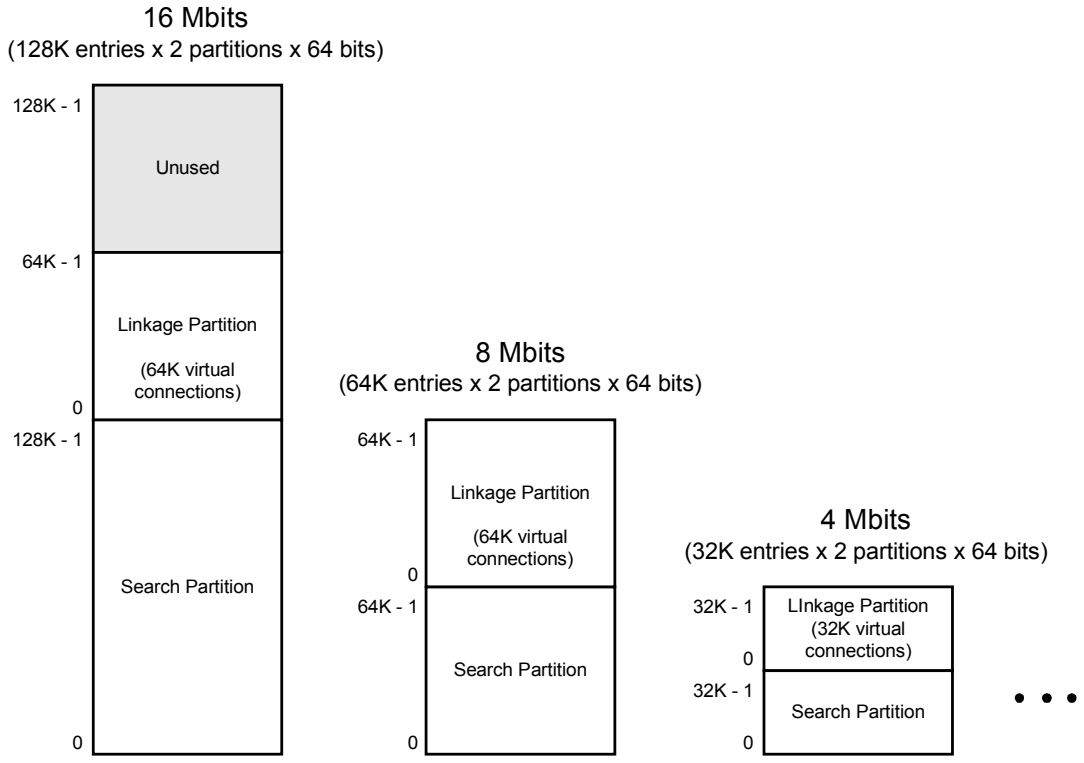
Figure 5 - Miscellaneous Data Structures



The size of the external SRAM can be varied to suit application specific requirements. The SRAM is always divided into a Linkage Partition and a Search Partition, which can support a maximum of 64K Linkage entries and 128K Search entries. In Figure 3, the SRAM shown is 8 Mbits (64K entries x 2 partitions x 64 bits). The partitioning configuration for various SRAM sizes is described below and shown in Figure 6.

- An 8 Mbit SRAM is the minimum size required to allow a full set of 64K virtual connections (one VC for each VC Linkage Table Record), and provides 64K Search entries.
- A 16 Mbit SRAM also allows 64K virtual connections but increases the number of Search entries to the maximum of 128K. The additional size of the Search Partition allows the full VCI-VPI-PHYID address to be resolved and eases the maintenance of the search table by providing extra room for modifications.
- Any SRAM smaller than 8 Mbits may be used with the number of allowed virtual connections decreasing proportionally.

Figure 6 - External SRAM Partitioning



When accesses are made to the SRAM through indirect registers, the microprocessor specifies which partition it wishes to access, and the address within the partition beginning at an offset of 0. The use of these partitions relies on the hardware connection between the MSB of the SRAM address bus and the MSB of the ATLAS-3200 SRAM address bus (SADDR[17]).

5 MICROPROCESSOR INTERFACE

The ATLAS-3200 has a generic microprocessor programming interface that provides access to a set of registers through the microprocessor interface bus. All microprocessor control and data transfer operations are performed through these registers. The sections below discuss the signals on the microprocessor interface bus and the direct registers.

5.1 Microprocessor Interface Bus

Microprocessor accesses to the ATLAS-3200 involve the 11-bit address bus (UP_ADDR[11:0]) that selects a register to be accessed and the 32-bit bi-directional data bus (UP_DAT[31:0]) over which data is transferred.

The busy signal (UP_BUSYB) is asserted while a microprocessor initiated access to external or internal RAM data is pending. The microprocessor can use this signal for polling or interrupt driven waiting. BUSY bits corresponding to each memory block are also accessible in the internal registers.

The DMA request signal (UP_DMAREQ) is asserted when the Output Microprocessor Cell Interface (OMCIF) contains a cell to be read. This signal can be polled by the microprocessor to check for arriving cells at the OMCIF. The interrupt signal (UP_INTB) can also be configured to indicate cell arrivals if an interrupt driven approach is used.

The interrupt request signal (UP_INTB) indicates when an unmasked interrupt source is active. The Interrupt Status registers must be read in order to identify the specific interrupt request source.

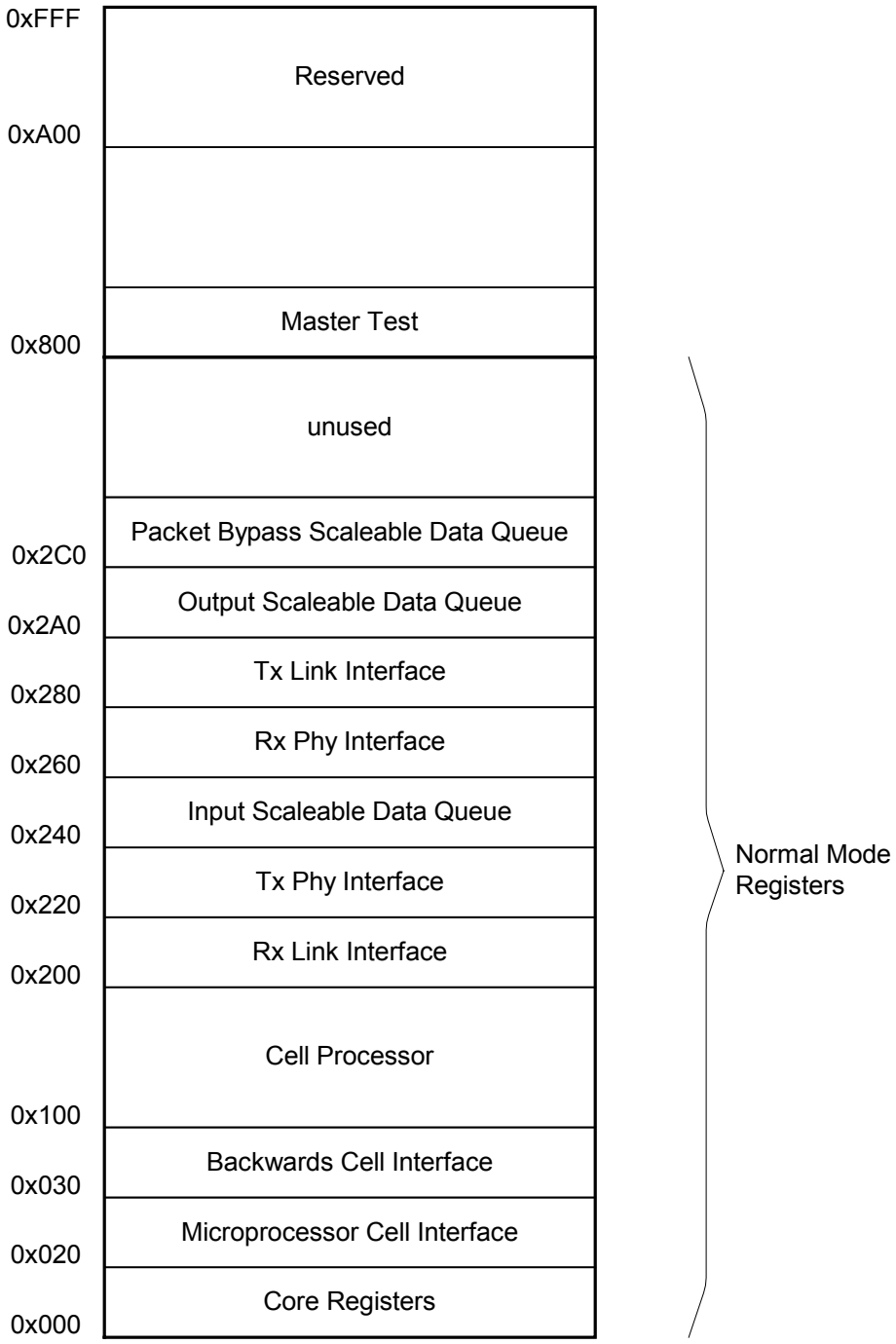
The chip select (UP_CSB), write (UP_WRB), read (UP_RDB) and address strobe (UP_ALE) signals are used for standard asynchronous read and write cycles as described in the Data Sheet [1].

The reset signal (UP_RSTB) performs a hard reset to the ATLAS-3200 when it is set low.

5.2 Register Memory Map

A memory map of the direct registers that are available to the microprocessor are listed in Figure 7. During typical operation only the Normal Mode Registers will be accessed. See the Data Sheet[1] for a comprehensive list of the Normal Mode Registers.

Figure 7 - Register Map



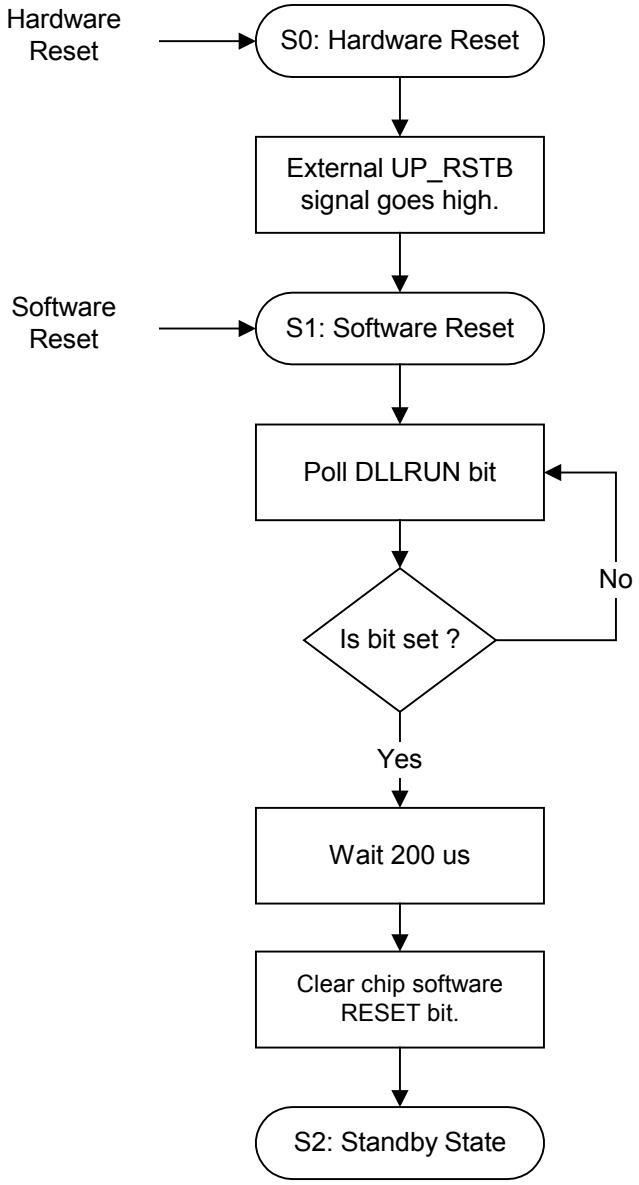
6 PROGRAMMING HIGH LEVEL OPERATIONS

This section contains guides to programming the major functional operations that will typically be performed on the ATLAS-3200. Programming algorithms, register configurations, explanations ATLAS-3200 operations, and some example pseudo-code are provided.

6.1 ATLAS-3200 Resetting

This section describes the ATLAS-3200's state transitions between the Hardware Reset, Software Reset and Standby states. A summary of the reset state transitions is shown in Figure 8. The algorithms to perform hardware and software resets are discussed in the sections below.

Figure 8 - Reset Flow Chart



Hardware Reset State:

The hardware reset state is entered when the UP_RSTB input pin is forced low. This state has the following characteristics:

- Internal registers (0x000 – 0xFFF) are reset to their default states.
- Digital outputs are tri-stated.
- All internal ATLAS-3200 components are in low-power, stand-by mode.

Software Reset State:

The Software Reset state is automatically entered after a hardware reset is removed or it can be entered by setting the RESET bit in the Master Reset and Configuration Register (0x000). This state has the following characteristics:

- Internal registers (0x000 – 0xFFF) are reset to their default states.
- Digital outputs are NOT tri-stated. The ATLAS-3200 is in Ingress Pos-PHY that is the state in which all UL3/POS pins that can be either inputs or outputs, are inputs. This avoids contention on startup.
- Generic Microprocessor Programming Interface and the DLL Clock are operational. All other internal ATLAS-3200 components in low-power, stand-by mode.

6.1.1 Algorithm for Hardware Resetting

The following steps need to be taken to proceed to the Software Reset state. The procedure to perform these steps will rely on the specific hardware implementation and may not be under microprocessor control.

1. Set up external signals. Ensure that the JTAG test pins are connected as listed in Table 1

Table 1 - Test Pin Connections

Pin Name	Value
TCK	0
TMS	0
TDI	0
TRSTB	1

2. Take the ATLAS-3200 out of hardware reset by forcing the UP_RSTB pin high.

6.1.2 Algorithm for Software Resetting

The following steps need to be taken to proceed to the Standby state. These steps are performed by the microprocessor through the Generic Microprocessor Programming Interface.

1. If this state was entered from the Hardware Reset state then leave the RESET bit in the Master Reset and Configuration Register (0x000) set to logic 1, and wait for the DLLRUN bit in the Master Clock Monitor Register (0x006) to be set to logic 1 for at least 200 us. This ensures that the SYSCLK DLL has been locked for sufficient time to allow the embedded DRAM to initialize.

2. Take the ATLAS-3200 out of software reset by writing a 0 to the RESET bit in the Master Reset and Configuration Register (0x000). Note that each of the components will still be held in reset by their individual RESET bits.

6.1.3 Example Routines

6.1.3.1 resetDevice

This function forces a software reset then brings the ATLAS-3200 into its standby state.

Inputs: (none)

Outputs: (none)

Pseudocode:

```
#define BTMSK_MASTER_CFG_AND_RST_RESET     0x0001
#define BTMSK_MASTER_CLOCK_MONITOR_DLLRUN  0x0080
#define BTMSK_IBCIF_CONFIG_IBCIFRST        0x0001

VOID resetDevice()
{

/*set RESET bit in Master Configuration and Reset Register*/
regWriteMask(REG_MASTER_CFG_AND_RST,
             BTMSK_MASTER_CFG_AND_RST_RESET,
```

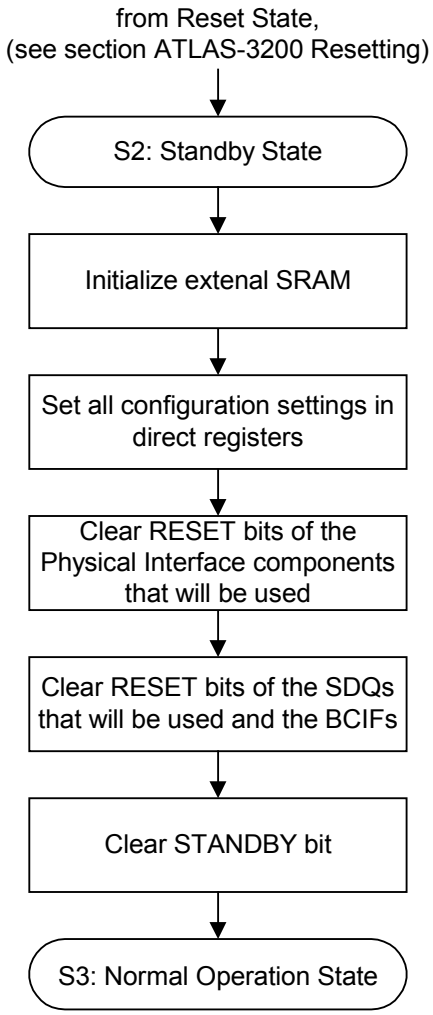
```
                BTMSK_MASTER_CFG_AND_RST_RESET)

/* wait until DLLRUN = 1 */
regPollBitHigh(REG_MASTER_CLOCK_MONITOR,
                 BTMSK_MASTER_CLOCK_MONITOR_ DLLRUN)


wait for 200us

/*clear RESET bit in Master Configuration and Reset
  Register*/
regWriteMask(REG_MASTER_CFG_AND_RST,
                BTMSK_MASTER_CFG_AND_RST_RESET, 0)

return
}
```

6.2 ATLAS-3200 Initializing

This section is a guide to programming the initialization routines for the ATLAS-3200. Initialization will typically be performed once after a reset. A summary of the initialization state transitions is shown in Figure 9.

Initializing the ATLAS-3200 involves setting the operational parameters for all the internal components to prepare them for Normal Operation state. The initialization should be performed while in Standby sate. Once initialization is complete the ATLAS-3200 can be brought into Normal Operation state. The sections below discuss the initialization algorithm and descriptions of all of the component settings that need to be initialized.

Figure 9 - Initialization Flow Chart



Standby State:

The Standby state is automatically entered after a software reset is removed or it can be entered from the Normal Operation state by setting the STANDBY bit in the Master Reset and Configuration Register (0x000).

The Standby state has the following characteristics:

- Cell Processor is disabled. The processing of cells currently in the pipeline is completed, but no more cells are accepted. This prevents passing corrupted cells while initializing the ATLAS-3200.
- All ATLAS-3200 components except the Cell Processor are enabled.
- All internal bus cycles are available for external SRAM and internal DRAM access (i.e. access to the Search Tables, VC Linkage Table and VC Table is given highest priority and no other processing will interrupt the SRAM and DRAM busses). The typical access times are given in Table 2.

Table 2 - Typical Memory Access Times in Standby State

Memory Location	Standby State	Normal Operation State
External SRAM	5 SYSCLK cycles	22 SYSCLK cycles
Internal DRAM	25 SYSCLK cycles	220 SYSCLK cycles

6.2.1 Algorithm

The algorithm to initialize the ATLAS-3200 is as follows:

1. Ensure the ATLAS-3200 is in Standby state by checking that the STANDBY bit in the Master Reset and Configuration Register (0x000) is set to logic 1.
2. ***IMPORTANT:*** Write logic 1 to both the SRAM_BUSY_EN bit and DRAM_BUSY_EN bit in the Master Reset and Configuration Register (0x000). These bits **must be set to 1** to enable the BUSY bits that are used to access all the indirect data structures. The default values of both the SRAM_BUSY_EN bit and DRAM_BUSY_EN are logic 0, so they must be changed after each reset.
3. Clear the external SRAM by writing zeroes to all memory locations.

4. Configure the global settings for all the internal components. The following sections provide the detailed setting descriptions for each component. Note that only the direct registers are accessable now since the components still have their individual reset bits set following the chip reset.

5. Clear the reset bits for the two Physical Interface components that will be used in the desired operating mode. The different operating modes are shown in Figure 2. See Table 3 for a list of the physical interface reset bits.

Table 3 - Register Bits for Physical Interface Resetting

Bit Name	Register	Operating Mode
RXLRST	RxL Configuration (0x200)	Ingress
RXPRST	RxP Configuration (0x260)	Ingress
TXLRST	TxL Configuration (0x280)	Egress
TXPRST	TxP Configuration (0x220)	Egress

6. Clear the reset bits for the SDQ components that will be used in the desired operating mode and the Backwards Cell Interface. The different operating modes are shown in Figure 2. See Table 4 for list of the reset bits. It is important that the SDQs are brought out of reset **after** the physical interfaces have been configured. This is because the Pre/Post Lengths in the physical in configuration registers propagate to the SDQs when they are activated.

Table 4 - Register Bits for SDQ and BCIF Resetting

Bit Name	Register	Operating Mode
SDQRST	Input SDQ Control (0x240)	Pos-Phy, Utopia
SDQRST	Output SDQ Configuration (0x2A0)	Pos-Phy, Utopia
SDQRST	Bypass SDQ Configuration (0x2C0)	Pos-Phy
IBCIFRST	Input Backwards Cell Interface Configuration (0x030)	(all)
OBCIFRST	Output Backwards Cell Interface Configuration (0x038)	(all)

7. Bring the ATLAS-3200 from Standby state into Normal Operation state by writing a logic 0 to the STANDBY bit in the Master Reset and Configuration Register (0x000).

6.2.2 Clock Settings

During the Reset state any changes to the default Delay Locked Loop clock configurations should have been made (see section 6.1 ATLAS-3200 Resetting). The remaining clock to configure is the Half Second Clock which is used as a trigger for OAM alarm monitoring. This is done by setting the GEN_HALFSECCLK bit in the Cell Processing Configuration Register (0x100) according to Table 5.

Table 5 - Register Bits for Half Sec. Clock Configuration

Bit Name	Register	Description
GEN_HALFSECCLK	Cell Processing Configuration Register (0x100)	0 : External signal HALFSECCLK is used. 1 : Clock is derived internally from SYSCLK. HALFSECCLK input is unused.

To check that all the clocks are running use the following procedure:

1. Read the Master Clock Monitor Register (0x006). This will clear the clock active bits.

2. Wait longer than the length of the longest clock period which is 500ms of the HALFSECCLOCK

3. Read the Master Clock Monitor Register (0x006) and check that all of the clock active bits have been set. This will confirm that the clocks are functioning.

6.2.3 Operation Mode Settings

After a reset the ATLAS-3200 defaults to the Ingress, Pos-PHY mode which is the state in which all POS/UL3 pins that can be either inputs or outputs, are inputs. This avoids external bus contention on startup.

Select the desired operation mode according to Table 6.

Table 6 - Register Bits for Operation Mode Settings

Bit Name	Register	Description	
Egress_IngressB	Master Configuration and Reset (0x000)	0	Ingress mode. RxLink and RxPHY blocks are used.
		1	Egress mode. TxPHY and TxLink blocks are used.
POS_UL3B	Master Configuration and Reset (0x000)	0	Utopia Level 3 signaling.
		1	POS-PHY 3 signaling.

6.2.4 Physical Interface Settings

The operation of the ATLAS-3200's physical interface depends on which operating mode it is in: Ingress/Egress and POS-PHY/Utopia. The registers that need to be programmed for each operating mode are shown shaded in Table 7. Unused registers do not need to be programmed.

Table 7 - Registers for Physical Interface Mode Config.

Register	Address	Ingress		Egress		Description
		POS-PHY	Utopia	POS-PHY	Utopia	
RxL Configuration	0x200					Reset bit and configuration settings
RxL Data Type Field	0x20C					Identification words used to identify incoming data as either ATM cells or packets.
RxL Interrupt Enable	0x201					Interrupt enables for the RxL Interrupt Register (0x202).
RxP Configuration	0x260					Reset bit and configuration settings.
RxP Data Type Field	0x266					Identification words inserted to outgoing ATM cells or packets.
RxP Interrupt Enable	0x262					Interrupt enables for the RxP Interrupt Register (0x261).
TxP Configuration	0x220					Reset bit and configuration settings
TxP Data Type Field	0x223					Identification words used to identify incoming data as either ATM cells or packets.
TxP Interrupt Enable	0x222					Interrupt enables for the TxP Interrupt Register (0x221).
TxL Configuration	0x280					Reset bit and configuration settings
TxL Data Type Field	0x28C					Identification words inserted to outgoing ATM cells or packets.
TxL Interrupt Enable	0x281					Interrupt enables for the TxL Interrupt Register (0x282).

The RxL, RxP, TxP, and TxL interfaces each have a reset bit in their Configuration registers. Before programming any of the configuration settings for a given interface, the interface's reset bit must be set. Once all the registers have been programmed the reset bit can be cleared.

In each Physical Interface mode the ATLAS-3200 supports 48 physical devices. The ATLAS-3200 may be configured to pass some physical connections transparently. This feature is typically used when cells from a certain PHY are to be processed in another device, such as another ATLAS-3200. This allows multiple Atlas-3200s to be cascaded, each handling cells from some of the PHY's. By default after a reset all physical connections will be processed. If it is desired that some physical connections not be processed, set the registers listed in Table 8 as required.

Table 8 - Registers for Physical Connection Processing

Register	Address	Description
Per-PHY Processing Enable 1	0x105	Process enable bits for physical connections 0 to 31.
Per-PHY Processing Enable 2	0x106	Process enable bits for physical connections 31 to 47.

6.2.5 Backwards Cell Interface Settings

The configuration of the Backwards Cell Interface is contained in the registers listed in Table 9. See the Data Sheet [1] for detailed descriptions of these registers. When programming the registers, the reset bits in the corresponding Configuration register must first be set before proceeding. Once the registers have been programmed the reset bits can be cleared.

Table 9 - Registers for BCIF Configuration

Register	Address	Description
Input Backwards Cell Interface Configuration	0x030	IBCIF reset bit and configuration settings.
Output Backwards Cell Interface Configuration	0x038	OBCIF reset bit and configuration settings.
Backward Cell Interface Pacing and Head of Line Blocking	0x104	Cell interval and timeout settings. See Data Sheet [1] for details.

6.2.6 Cell Processor Settings

The general operation of the cell processor is controlled by the Cell Processor Configuration Register (0x100) that contains the configuration bits listed in Table 10. For detailed descriptions of these settings see the Data Sheet [1] and other sections of this Programming Guide referenced in Table 10.

Table 10 - Cell Processor Config. Register (0x100)

Bit	Name	Description
31:29	Unused	Always set to logic 0 when writing. Read value is undefined.

Bit	Name	Description
28	Copy_FwPM_Timestamp	(see 6.11.3 PM Cell Processing)
27	GEN_HALFSECCLK	(see section 6.2.2)
26	F4SAISF5EAIS	(see 6.11.2 Fault Management Cell Processing)
25	F4SAISF5ERDI	
24	F4EAISF5EAIS	
23	F4EAISF5SRDI	
22	ForceCC	
21	AUTO_AIS	
20	COS_DRAM_ERR_EN	(see Section 6.9.1 Change of State FIFO)
19	Reserved	Always set to logic 0 when writing. Read value is undefined.
18	COS_Fail_EN	(see Section 6.9.1 Change of State FIFO)
17	COS_FAIL_ONLY	
16	COS_EN	
15	Sat_Fast_PM_Counts	(see Section 6.9.2 Count Rollover FIFO.)
14	CRO_FIFO_EN	
13	Alternate_Count	0 : Arriving cells increment Cell Count fields in all VC Table Records 1 : Arriving cells increment Alternate Count fields in all VC Table Records.
12	VP_RM_PTI6	0 : VPC (F4) Resource Management cells identified by VCI=6 and PTI is ignored. 1 : VPC (F4) Resource Management cells identified by VCI=6 and PTI=110.
11	Search_Verify_En	(see 6.7 Virtual Connection Search Tree Modifying)
9	SRAM_Even_Parity	(see Section 7.2 External SRAM)

Bit	Name	Description
8	Cell_Info_to_OCIF	Enables insertion of Cell Information into outgoing cells routed to the OCIF. This information is for the convenience of an ASIC or tester, to assist in identifying the cell and the connection from which it came
7	Timeout_To_UP	Controls routing of cells that timeout while waiting to be inserted from the IBCIF. See Section 0 for details.
6	Reserved	Always set to logic 0 when writing. Read value is undefined.
5	Cell_Info_To_UP	Enables insertion of Cell Information into cells routed to the MCIF. This information is to assist in identifying the cell type and the reason why it was routed to the MCIF.
4	XGFC	Controls insertion of translated fields into outgoing cells.
3	XUDF	
2	XHEC	
1	XPREPO	
0	XVPIVCI	

6.2.7 Cell Counting Settings

The ATLAS-3200 supports per-VC and per-PHY cell counting. The cell counting configuration registers are listed in Table 11. These registers should typically be set during initialization, however, they can also be re-configured during Normal Operation state.

The per-VC cell counts are stored in fields in the VC Table Records. The Cell Counting Configuration Register specifies on a global basis, which type of cells are counted in the per-VC counts. Program the Cell Counting Configuration Register as required for the application.

The per-PHY cell counts are stored in Per-PHY Counter Memory (see Figure 5) and are accessed through indirect registers. The Per-PHY Counter Configuration Register specifies on a global basis which type of cells are counted in the per-PHY counts. Program the Per-PHY Counter Configuration Register as required for the application.

Table 11 - Registers for Cell Counting Configuration

Register	Address	Description
Cell Counting Configuration Register	(0x102)	Specifies the type of cells counted in per-VC counts. See the Data Sheet [1] for details.
Per-PHY Counter Configuration Register	(0x1A0)	Specifies the type of cells counted in per-PHY counts. See the Data Sheet for details.

6.2.8 Policing Settings

The ATLAS-3200 supports per-VC and per-PHY transmission rate policing. The policing configuration registers are listed in Table 12. These registers should typically be set during initialization, however, they can also be re-configured during Normal Operation state. See the Data Sheet [1] for detailed register descriptions, and see Section 6.3.3 and Section 6.5.7 for per-PHY and per-VC policing information respectively.

ATM cell flows are policed using the Generic Cell Rate Algorithm (GCRA) and packet flows are policed using the Guaranteed Frame Rate (GFR) algorithm.

The ATLAS-3200 has eight pairs of configurations that the per-VC policers select from. Each VC can have up to two per-VC GCRA policing operations active.

The ATLAS-3200 has four global configurations the per-PHY policers select from. Each PHY connection can optionally have one per-PHY GCRA policing operation active.

Table 12 - Registers for Policing Configuration

Register	Address	Description
Per-VC Non-Compliant Cell Counting Configuration	0x130	Defines the VC non-compliant cells.
Connection Policing Configuration 1 & 2	0x131	VC GCRA configurations 1 & 2
Connection Policing Configuration 3 & 4	0x132	VC GCRA configurations 3 & 4

Register	Address	Description
Connection Policing Configuration 5 & 6	0x133	VC GCRA configurations 5 & 6
Connection Policing Configuration 7 & 8	0x134	VC GCRA configurations 7 & 8
PHY Policing Configuration	0x142	PHY GCRA configurations 1, 2, 3 & 4
Per-PHY Non-Compliant Cell Counting Configuration	0x143	Defines the PHY non-compliant cells.

6.2.9 OAM Settings

Operations and Maintenance (OAM) operation is controlled by global configuration settings, per-VC configuration settings, and per-Performance Management settings. During initialization or at run-time the global OAM configurations can be set to their desired values. See Section 6.8 OAM Cell Processing for details on the global OAM configuration settings.

6.2.10 Search Key Settings

The Search Key settings that should be configured are contained in the register listed in Table 13. See section 6.7 Virtual Connection Search Tree Modifying for further information.

Table 13 - Registers for Search Key Configuration

Register	Address	Description
Search Engine Configuration Register	0x10B	Specifies construction of the Primary and Secondary Search keys.

6.2.11 Interrupt Settings

After a reset all interrupts are disabled by default. While the ATLAS-3200 is in standby state no interrupts will be generated. Enable the desired interrupts while in Standby state and once the Normal Operation state is entered they will become active. See section 6.10 Interrupt Handling for interrupt information.

6.2.12 VC Table Settings

The VC Table can be modified during Normal Operation state as connections are added and removed, however the VC Table Maximum Index Register (0x110) is a control register that should be set during initialization. This register holds the maximum VC Table address (VCRA[15:0]) and is used by the background processes of the VC Table. This should be set to the maximum address supported by the external SRAM. To ensure proper operation it is important that the VC Table Maximum Index Register not be set to an address greater than that supported by the external SRAM.

***NOTE**: For proper OAM processing the VC Table Maximum Index Register must never be set to zero.*

6.3 Physical Connection Adding

This section is a guide to programming the routines to add a physical connection to the ATLAS-3200. After the ATLAS-3200 has been initialized, the routines to add physical connections will typically be run once to establish connections to the attached hardware. A physical connection can either be configured to transmit ATM cells or packets.

Adding a physical connection involves configuring four components: Physical ID Mapping, Per-PHY Policing, Scalable Data Queues, and Polling and Servicing Calendars. The general programming algorithm and descriptions of the configuration settings for each of those components are discussed in the sections below.

6.3.1 Algorithm

The algorithm to add a physical connection for **ATM cell transmission** is as follows:

1. Ensure that this PHY's ProcessPHY bit in the Per-PHY Processing Enable Register (0x105 or 0x106) is set to logic 1.

2. Configure the physical ID mapping. (See Section 6.3.2 for settings.)

3. Set the per-PHY policing options. (See Section 6.3.3 for settings.)

4. Disable and flush the physical connection's FIFO in the Bypass SDQ. To do this perform an indirect write to the PHY's Bypass SDQ entry with the ENABLE bit in the Bypass SDQ Indirect Configuration Register set to logic 0 and the FLUSH bit set to logic 1

5. Activate the physical connection's FIFO in the Output SDQ by allocating the location and size of the FIFO. (See Section 6.3.4 for settings.)

6. If the output interface uses a polling and servicing calendar in the current operating mode, update the appropriate Polling and Servicing Calendar to include this physical ID. (See Section 6.3.5 for settings.)

7. Activate the physical connection's FIFO in the Input SDQ by allocating the location and size of the FIFO. (See Section 6.3.4 for settings.)

8. If the input interface uses a polling and servicing calendar in the current operating mode, update the appropriate Polling and Servicing Calendar to

include this physical ID. The Calendars should only be changed while the Cell Processor is in STANDBY mode. (See Section 6.3.5 for settings.)

The algorithm to add a physical connection for **packet transmission** is as follows:

1. Configure the physical ID mapping. (See Section 6.3.2 for settings.)

2. Disable the physical connection's FIFOs in the Input and Output SDQs. To do this perform indirect writes to the PHY's Input and Output SDQ entries with the ENABLE bit in the Input and Output SDQ Indirect Configuration Registers set to logic 0 and the FLUSH bit set to logic 1. (See Section 6.3.4 for settings.)

3. Activate the physical connection's FIFO in the Bypass SDQ by allocating the location and size of the FIFO. (See Section 6.3.4 for settings.)

4. If the output interface uses a polling and servicing calendar in the current operating mode, update the appropriate Polling and Servicing Calendar to include this physical ID. The Calendars should only be changed while the Cell Processor is in STANDBY mode. (See Section 6.3.5 for settings.)

6.3.2 Physical ID Mapping Settings

The RxLink and TxLink blocks can remap the physcial ID's (PHY IDs) of the incoming data flows to arbitrary PHY IDs. The remapped PHY IDs are used by all downstream ATLAS-3200 internal components and external devices. See Figure 2 to determine the downstream components from the RxLink and TxLink blocks. The PHY ID mapping tables contain one entry for each of the 48 possible physical devices and are accessed through indirect registers. By default after a reset each entry maps the external PHY ID to its identical PHY ID.

See section 7.6 PHY ID Mapping Table Interfacing for details on programming the PHY ID mapping tables using the indirect access registers.

6.3.3 Per-PHY Policing Settings

The ATLAS-3200 supports one GCRA policing operation on each of the fourty eight physical connections (PHYs). Per-PHY policing is active for a given PHY when its corresponding bit is set in its PHY Policing Enable register. Per-PHY policing is performed on the cumulative cell flow of all VC's on the PHY that have the PHY Police bit in the Policing Configuration field of their VC Table Record set. For each VC, the PHYID[5:0] field in its VC Linkage Table Record determines which of the 48 PHY policing instances is addressed.

The PHY policing operations share four global GCRA configurations in the PHY Policing Configuration Register (0x142). When policing is enabled for a given PHY one of the four per-PHY GCRA configurations must be selected.

The policing configuration for each PHY is stored in internal RAM that is programmed indirectly through the PHY Policing Address Access Control Register and the PHY Policing RAM Data registers. See Section 7.7 PHY Polcing RAM Interfacing for the programming algorithms using the indirect registers.

All the registers related to per-PHY policing are listed in Table 14.

Table 14 - Registers for Per-PHY Policing Configuration

Register	Address	Description
PHY Policing Enable 1	0x140	Policing enable bits for physical connections 0 to 31.
PHY Policing Enable 2	0x141	Policing enable bits for physical connections 31 to 47.
PHY Policing Configuration	0x142	PHY GCRA configurations 1,2,3, and 4. Each PHY policing selects one of these configurations. This will typically be set during initialization.
Per-PHY Non-Compliant Cell Counting Configuration	0x143	Defines the PHY non-compliant cells. This will typically be set during initialization.
PHY Policing RAM Address and Access Control	0x144	Indirect access registers for accessing the fourty eight PHY Policing RAM tables. See Section 7.7 PHY Polcing RAM Interfacing for programming algorithm
PHY Policing RAM Data Row 0	0x145	
PHY Policing RAM Data Row 1	0x146	
PHY Policing RAM Data Row 2	0x147	
PHY Policing RAM Data Row 3	0x148	

The procedure to **disable** per-PHY policing for a physical connection is as follows:

1. Write the corresponding enable bit to 0 in either the PHY Policing Enable 1 Register or the PHY Policing Enable 2 Register.

The procedure to **enable** per-PHY policing for a physical connection is as follows:

1. Write the PHY Policing RAM entry for this PHY.

    - Select appropriate values for the PHY I (Increment), PHY L (Limit), Phy Police Config, PHY Policing Rollver FIFO EN, PHY VC Count and Phy Action fields. See the Cell Rate Policing section in the Data Sheet [1] for detailed description of the calculations used to determine these values.

    - Write zeroes to the Phy TAT LSB, Reserved, Phy TAT MSB, Phy Non-Compliant1, Non-Compliant2, and Non-Compliant3 fields.

    See Section 7.7 PHY Polcing RAM Interfacing for the indirect write programming algorithm.

2. Write the corresponding enable bit to logic 1 in either the PHY Policing Enable 1 Register or the PHY Policing Enable 2 Register.

6.3.4 Scaleable Data Queue Settings

There are three Scaleable Data Queues (SDQs) in the ATLAS-3200. Physical connections transmit ATM cells using the Input SDQ and Output SDQ or transmit packets using the Bypass SDQ. When the ATLAS-3200 is in Utopia mode only cell transmission through the Input and Output SDQs is allowed. When the ATLAS-3200 is in POS_PHY mode, each physical connection can either transmit cells through the Input and Output SDQs or transmit packets through the Bypass SDQ.

Each SDQ has 12288 bytes (192 ATM cells) of memory that is divided into 96 blocks. FIFOs are allocated within the SDQs by specifying a starting block address and the size in blocks. The number of blocks that should be allocated to a FIFO depend on the physical connection's bandwidth. Suggested minimum FIFO sizes are given in Table 15.

Table 15 - Suggested SDQ FIFO Sizes

Bandwidth	FIFO size (blocks)	FIFO size (cells)	FIFO size (bytes)
Below STS-1	1	2	128
STS-1 or less	2	4	256
STS-3	6	12	768
STS-12 or STS-48	24	48	3072

Bandwidth	FIFO size (blocks)	FIFO size (cells)	FIFO size (bytes)
STS-48	96	192	12288

To achieve optimum usage of the SDQs, the FIFOs should be allocated with consideration given to the following:

- matching the length of the FIFO to the physical connection's bandwidth
- aligning the FIFOs efficiently reduce unused memory

The operation of the SDQs is discussed in detail in the Operations sections of the Data Sheet [1]. Refer to this section for the complete set of rules governing assigning FIFOs in the SDQs. The registers related to the SDQs are listed in Table 16.

Table 16 - SDQ Registers

Input SDQ Registers	Output SDQ Registers	Bypass SDQ Registers	Description
Input SDQ Control (0x240)	Output SDQ Control (0x2A0)	Bypass SDQ Control (0x2C0)	Reset and Transfer In Progress bits.
Input SDQ Interrupts (0x241)	Output SDQ Interrupts (0x2A1)	Bypass SDQ Interrupts (0x2C1)	Interrupt status and enable bits, for per-FIFO error conditions.
Input SDQ Interrupt ID (0x242)	Output SDQ Interrupt ID 0x2A2)	Bypass SDQ Interrupt ID 0x2C2)	Fields that identify the FIFO that caused an interrupt.
Input SDQ Indirect Address (0x244)	Output SDQ Indirect Address (0x2A4)	Bypass SDQ Indirect Address (0x2C4)	Indirect access registers. See Section 7.4 SDQ Entry Interfacing for details.
Input SDQ Indirect Configuration (0x245)	Output SDQ Indirect Configuration (0x2A5)	Bypass SDQ Indirect Configuration (0x2C5)	
Input SDQ Indirect Cells and Packets Count (0x246)	Output SDQ Indirect Cells and Packets Count (0x2A6)	Bypass SDQ Indirect Cells and Packets Count (0x2C6)	
Input SDQ Cells Accepted Aggregate Count (0x247)	Output SDQ Cells Accepted Aggregate Count (0x2A7)	Bypass SDQ Cells Accepted Aggregate Count (0x2C7)	Aggregate count of all the ATM cells accepted by the Output SDQ. Ensure that the TIP bit in the SDQ Control register is logic 0 before reading this register.
Input SDQ Cells Dropped Aggregate Count (0x248)	Output SDQ Cells Dropped Aggregate Count (0x2A8)	Bypass SDQ Cells Dropped Aggregate Count (0x2C8)	Aggregate count of all the ATM cells dropped by the Output SDQ due to overflow or transfer errors. Ensure that the TIP bit in the SDQ Control register is logic 0 before reading this register.

The procedure to program the SDQs involves using indirect registers to write and read from the SDQ memory. The basic SDQ read and write operations are discussed in Section 7.4 SDQ Entry Interfacing.

The algorithm to **enable** a FIFO in an SDQ is as follows:

1. Determine the current configuration of all FIFOs in the SDQ and locate the empty memory slots. Two general approaches can be used to determine the current SDQ configuration:

   - Read the SDQ Indirect Configuration Register for all 48 physical IDs, then sort the data into a memory map.

   - Store a copy of the SDQ configuration on the microprocessor. This method reduces the number of external accesses and allows the use of conveniently sorted data structures.

2. Determine the size of the FIFO that is required for the new physical connection. Suggested values are given in Table 15, but if it is known that there will be extra memory, larger values can be used.

3. Locate an empty memory slot of the required size using the current SDQ configuration information.

4. Wait for the current FIFO to empty or force a flush, then write the SDQ entry for the new FIFO with its ENABLE bit set to logic 1. See Section 7.4 SDQ Entry Interfacing for the indirect writing algorithm.

5. If a local microprocessor copy of the SDQ configuration is being maintained, update it to indicate the newly allocated memory blocks.

The algorithm to **disable** a FIFO in an SDQ is as follows:

1. Flush the current FIFO, then write the SDQ entry for the new FIFO with its ENABLE bit set to logic 0. See Section 7.4 SDQ Entry Interfacing for the indirect writing algorithm.

2. If a local microprocessor copy of the SDQ configuration is being maintained, update it to indicate the newly freed memory blocks.

6.3.5 Polling and Servicing Calendar Settings

The physical interfaces that are bus masters use a Polling and Servicing Calendar to perform weighted servicing of the PHYs. The exception to this is the

Ingress Utopia Rx Slave Interface which can present itself as a single PHY to the Rx Master and uses its calendar to perform internal polling and servicing of the PHYs.

There are three Polling and Servicing Calendars in the ATLAS-3200: one in the RxLink, the RxPHY and the TxLink block. The operating mode that the ATLAS-3200 is in determines which physical interfaces are active and thus which calendars are used. The calendar use in each of the operating modes is shown in Figure 2 and summarized in Table 17.

Table 17 - Polling and Servicing Calendar Use

Mode	ATLAS-3200 Input Interface		ATLAS-3200 Output Interface	
	Interface Type	Calendar Used	Interface Type	Calendar Used
Ingress Utopia	RxLink	YES	RxPHY	OPTIONAL
Ingress POS-PHY	RxLink	NO	RxPHY	YES
Egress Utopia	TxPHY	(none)	TxLink	YES
Egress POS-PHY	TxPHY	(none)	TxLink	YES

The Polling and Servicing Calendars are circular lists of PHY IDs that have a configurable length up to a maximum of 128. The more often a given PHY ID appears in the calendar, the more often it gets polled. The microprocessor is responsible for programming the calendar. The ordering of the PHY IDs within the calendars should follow the following principles:

- The number of entries that a given PHY has in a calendar should be proportional to its bandwidth with respect to the bandwidth of the other PHYs. Typically the lowest rate PHY will have one entry and all other PHYs will have a proportionately higher number of entries. (See the Data Sheet [1] for examples)

- The entries for a given PHY should be evenly distributed throughout the calendar.

See the examples in Section 6.3.5.1 for a demonstration of these pronciples.

6.3.5.1 Calendar Table Configuration Examples

Some example Calendar configurations are given below. For further information and an additional example see the Data Sheet [1].

With one connection, only one entry is needed reagardless of its speed as shown in Table 18.

Table 18 - Calendar Example: One STS-12

Calendar Address	Calendar Data
0	STS-12a PHY ID

With two connections that have a rate ratio of 4:1 (STS-12:STS-3), the number of entries of the faster connection should increase to the proportionate ratio. See Table 19.

Table 19 - Calendar Example: One STS-12, One STS-3

Calendar Address	Calendar Data
0	STS-12a PHY ID
1	STS-3a PHY ID
2	STS-12a PHY ID
3	STS-12a PHY ID
4	STS-12a PHY ID

In the case of a single fast connection and multiple slower connections, the number of entries for the faster connection still remains at the proportion of its rate with respect to the rate of the slowest connection (4:1), regardless of the actual number of connections. The entries for a given connection are also distrubuted as evenly as possible as shown in Table 20.

Table 20 - Calendar Example: One STS-12, Two STS-3's

Calendar Address	Calendar Data
0	STS-12a PHY ID
1	STS-3a PHY ID
2	STS-12a PHY ID
3	STS-12a PHY ID
4	STS-3b PHY ID
5	STS-12a PHY ID

6.3.5.2 Calendar Programming Procedure

The procedure to program the Calendars involves using the indirect registers listed in Table 21 to write and read from the Calendar memory. Generally, the calendar should be set up at device initialization and subsequently be left unchanged. When the calendar length, or a calendar entry is updated during cell or packet flow, there may be an impact on polling, which may result in loss of data for a short period of time on any PHY that is transferring data.

Table 21 - Registers for Calendar Programming

RxL Calendar Registers	RxP Calendar Registers	TxL Calendar Registers	Description
RxL Calendar Length (0x20A)	RxP Calendar Length (0x260)	TxL Calendar Length (0x28A)	Index of the last calendar entry. Once polling and servicing pointers increment to this value they loop back to 0.
RxL Calendar Indirect Address and Data (0x20B)	RxP Calendar Indirect Address and Data (0x265)	TxL Calendar Indirect Address and Data (0x28B)	Indirect address field, data field, busy bit and read write bit. Calendar entries programmed indirectly through this register.

The algorithm to update the calendar entries to incorporate a new PHY is as follows. These procedures apply identically to all of the Calendars unless noted otherwise.

1. Determine the current configuration of Calendar to identify what PHYs are currently in the Calendar. then determine the bandwidth of each PHY in the Calendar. Two approaches can be used to do this:

    - Read the Calendar entries from 0 up to the calendar length through the Calendar Indirect Address and Data register. Create a list of all the PHYs in the Calendar and determine the transmission speed of each one. A PHY's connection speed can be deduced indirectly by the length of its SDQ FIFO (depending on the FIFO allocation algorithm) but should likely be available from a data structure in the microprocessor.

    - Store a copy of the Calendar configuration on the microprocessor with the transmission speeds of each PHY connection. This method reduces the number of external accesses and allows the use of convenient data structures.

2. Create a new Calendar table incorporating entries for the new PHY using the guidelines discussed previously.

3. Write the updated Calendar entries through the Calendar Indirect Address and Data register. The Calendar should not be in active operation while its entries are being updated otherwise cells may be corrupted.

4. Write the updated calendar length to the Calendar Length Register value.

6.4 Physical Connection Removal

This section is a guide to programming the routines to remove a physical connection from the ATLAS-3200. Removing a physical connection will typically be done following a procedure that prevents data from being lost.

The algorithms to remove PHY's transmitting ATM cells and Packets are given in the following section. The four components used during this procedure are the Physical ID Mapping, Per-PHY Policing, Scalable Data Queue, and Polling and Servicing Calendar. Descriptions and programming algorithms for these components are discussed in section 6.3 Physical Connection Adding so they are not discussed again here.

6.4.1 Algorithm

The algorithm to remove a physical connection transmitting **ATM cells** is as follows:

1. Disable the physical connection's FIFO in the Input SDQ. To do this perform an indirect write to the FIFO entry with the ENABLE bit set to logic 0.

2. Disable the physical connection's FIFO in the Output SDQ. To do this perform an indirect write to the FIFO entry with the ENABLE bit set to logic 0.

3. Update the Polling and Servicing Calendars that are currently being used to remove all entries for this PHY. The Calendars should only be changed while the Cell Processor is in STANDBY mode.

The algorithm to remove a physical connection transmitting **packets** is as follows:

1. Disable the physical connection's FIFO in the Bypass SDQ. To do this perform an indirect write to the FIFO entry with the ENABLE bit set to logic 0.

2. Update the Polling and Servicing Calendars that are currently being used to remove all entries for this PHY. The Calendars should only be changed while the Cell Processor is in STANDBY.

6.5 Virtual Connection Adding

This section is a guide to programming the routines to add a virtual connection (VC) to the ATLAS-3200.

Each virtual connection is stored in the ATLAS-3200 as configuration settings in a VC Table Records and VC Linkage Table Records. All the VC Table Records and VC Linkage Table Records are organized in a VC Search Tree that allows them to be efficiently searched for. To add a new VC, a VC Table Record and a VC Linkage Table Record are configured and are then inserted into the VC Search Tree.

The ATLAS-3200 supports Virtual Channel Connections (VCCs) and Virtual Path Connections (VPCs). The VCCs carry F4 OAM flows and VPCs carry F5 OAM flows. Each VCC may be independent or it may be part of a VPC, and each.VPC can encapsulate multiple VCCs.

The sections below discuss the configuration of VC's for VCC or VPC operation, the general algorithm to add a VC, and then the specific VC Table Record configuration settings.

6.5.1 Adding an Independent VCC

To add a VCC that will operate independently, without an associated VPC, the only configuration requirement is:

- In its VC Linkage Table Record: VPC Pointer Active = logic 0

All F5 OAM flows may operate on this connection but no F4 flows will be processed.

6.5.2 Adding an Independent VPC

To configure a VPC to operate independently, without any explicit enclosed VCCs, the configuration requirements are:

- In its VC Linkage Table Record: VPC Pointer Active = logic 0
- In its VC Table Record: VCI = 0x0000

All F4 OAM flows may operate on this connection but no F5 OAM flows will be processed. F5 OAM cells will be treated like user cells.

Search Tree Requirements:

When the ATLAS-3200 matches incoming cells to the VPC's VC Table Record it ignores the cell's VCI value in the comparison. This is the special behaviour caused by setting the VCI field in the VC Record Table to 0. As a result, all cells which belong to this VPC can be correctly routed to the VPC record with only a single search tree entry. The requirements on the search tree to allow this to work are:

- The Selector field in the Secondary Search entries must decrease monotonically at each subsequent branch in the Search Tree.
- All VPC cells must resolve to the same primary search key.

See Section 6.7 Virtual Connection Search Tree Modifying for more information on the Search Tree.

6.5.3 Adding a VCC within a VPC

To add a VCC within a VPC, so that both F4 and F5 OAM flows may be used, the VC Table Records, VC Linkage Table Records, and the Search Tree must all meet specific configuratio requirements. The requirements for the VPC, the VCCs, and the Search tree are listed below, and shown in Figure 10 and Figure 11.

VPC (F4) requirements:

- In its VC Linkage Table Record: VPC Pointer Active = logic 0
- In its VC Table Record: VCI = 0x0000
- Its VC Table Record must be in a different memory Bank than the VC Table Records of all associated VCCs. There are 4 memory banks, interleaved by the 2 LSBs of the VC Record Addresses as shown in Figure 3. Therefore VCRA[1:0] of this VPC Table Record must not equal VCRA[1:0] for any of the VCC Table Records.

VCC (F5) Requirements::

- In each VC Linkage Table Record: VPC Pointer Active = logic 1
- In each VC Table Record: VPC Pointer = VCRA of the VPC Table Record.
- Each VCC Table Record must be in a different memory Bank than that of its associated VPC Table Record. There are 4 memory banks, interleaved by

the 2 LSBs of the VC Record Addresses as shown in Figure 3. Therefore VCRA[1:0] of this VCC Table Record must not equal VCRA[1:0] of the VPC Table Record.

Search Tree Requirements:

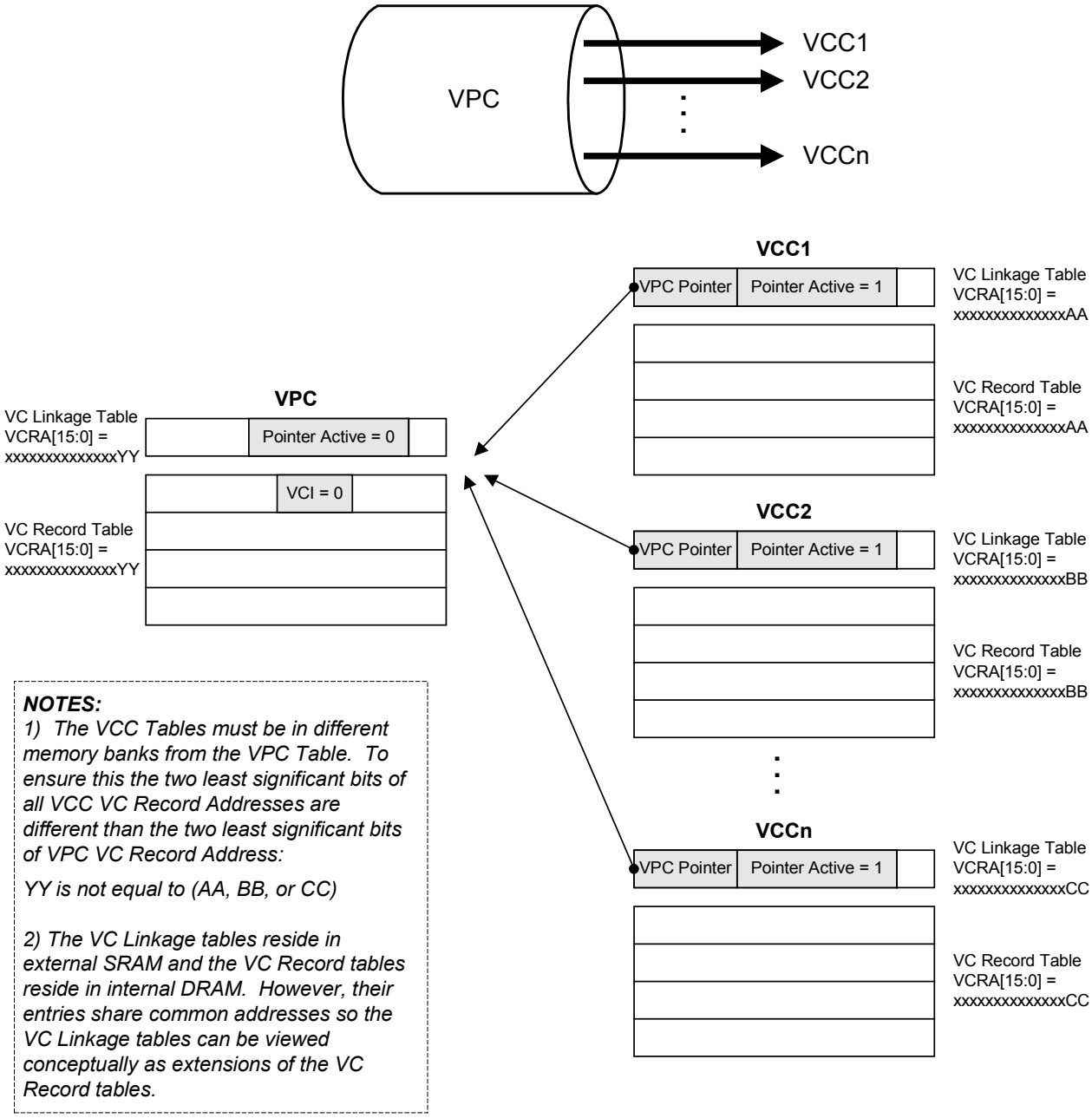
The same search tree requirements as listed in section 6.5.2 Adding an Independent VPC.

PM Requirements:

To configure an F4 PM session on the VPC, the VC Linkage Table Records of the VPC and its associated VCCs should be setup as follows:

- In the VPC VC Linkage Table Record: Set PM1_Address or PM2_Address to point to the allocated PM Table Record, and set PM1_Active or PM2_Active.
- In each VCC VC Linkage Table Record: Set PM1_Address/PM2_Address and PM1_Active/PM2_Active to point to the same PM Table Record as the VPC. The other PMx_Address/PMx_Active may be used to configure a PM session on only that VCC.

Figure 10 - VCC within VPC Configuration Requirements



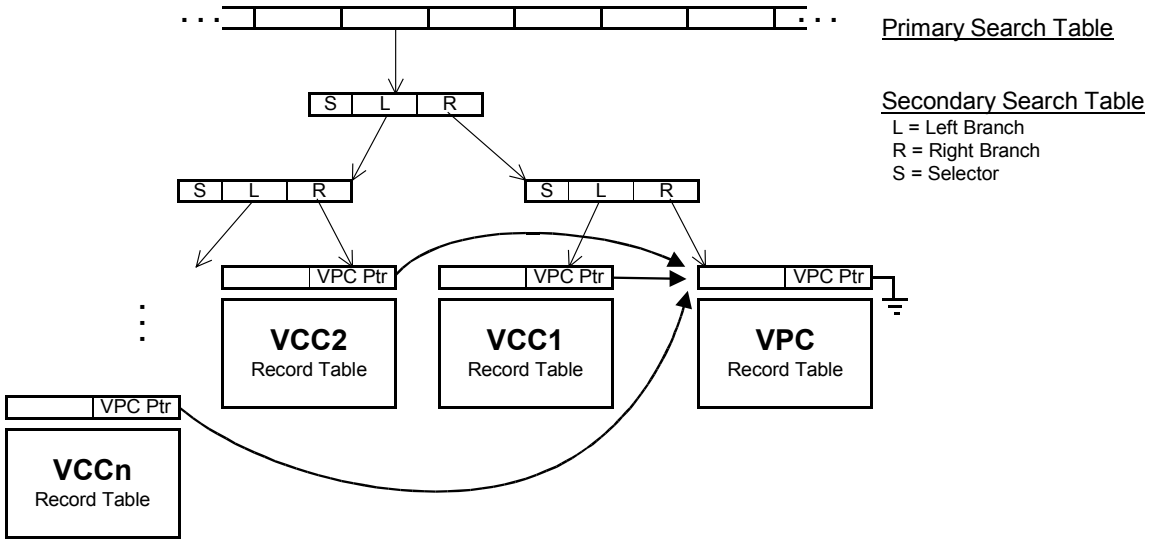
NOTES:

1) The VCC Tables must be in different memory banks from the VPC Table. To ensure this the two least significant bits of all VCC VC Record Addresses are different than the two least significant bits of VPC VC Record Address:

YY is not equal to (AA, BB, or CC)

2) The VC Linkage tables reside in external SRAM and the VC Record tables reside in internal DRAM. However, their entries share common addresses so the VC Linkage tables can be viewed conceptually as extensions of the VC Record tables.

Figure 11 - VCC within VPC Search Tree Example



6.5.4 Algorithm

1. Locate an available VC Record address (VCRA) and write a VC Table Record to that address with the desired Configuration, Addressing, Policing, and OAM settings. These settings are discussed in the sections below.

    - If this virtual connection is a VCC (F5 flow) that will be joining an existing VPC (F4 flow), ensure that its VCRA specifies a different memory bank than the bank that the VPC is in.

    - If this virtual connection is a VPC (F4 flow) ensure that the VCI field is set to 0x0000.

2. Optionally configure a new Performance Management Table. If a Performance Management session is desired for this connection either a new PM Table Record can be created or the VC can link to an existing PM Table Record. More than one VC can link to a single PM Table Record. See Section 6.8 for details.

3. Write the VC Linkage Table Record to the Linkage partition of external memory using the same address as the VC Table Record. Set the desired PHY ID, and optionally set links to PM Table Records.

- If this virtual connection is an independent VCC (F5 flow) ensure that VPC Pointer Active is set to logic 0.
- If this virtual connection is a VCC (F5 flow) that will be joining an existing VPC (F4 flow), set VPC Pointer Active to logic 1 and set the VPC Pointer to the VCRA of the VPC's VC Table Record.
- If this virtual connection is a VPC (F4 flow) ensure that VPC Pointer Active is set to logic 0.

4. Enable the VC by writing a logic 1 to the Active bit in the VC Table Record Configuration field.
5. Modify the Search Tables to insert a pointer to the new VC Table Record and VC Linkage Table Record into the search tree. See section 6.7 Virtual Connection Search Tree Modifying for details.

6.5.5 VC Table Configuration Settings

The Configuration field in a VC Table Record sets the VC's general configuration options. The bit settings are described in Table 22.

Table 22 - VC Table Record, Configuration Field

Bit	Name	Description
13	Reserved	Always set to logic 0 when writing. Read value is undefined.
12	Reserved	Always set to logic 0 when writing. Read value is undefined.
11	Active	This should be set to 0 while configuring the connection. When the connection is ready to be activated the last step should be to set this to 1.
10	NNI	0 : This connection point is a User Network Interface. 1 : This connection point is a Network-Network Interface.
9	Count Config Select	0 : Use Cfg0 configuration from the Cell Counting Configuration Register (0x102) 1 : Use Cfg1 configuration from the Cell Counting Configuration Register (0x102)
8	APStoUP	These bits set the per-VC routing options for this VC. See Section 6.12 Cell Routing for cell flow information.

Bit	Name	Description
7:6	LB_Route[1:0]	
5	FM_to_UP	
4	VC_to_BCIF	
3	VC_to_UP	
2	Drop_VC	
1	Rollover_FIFO_enable	Enables the cell count fields to generate CRO FIFO entries. See Section 6.9.2 Count Rollover FIFO for detailed CRO FIFO information. 0 : Cell Count and Alternate Cell Count fields will saturate at the maximum value. Use this setting if the Count Rollover FIFO is not being used. The Cell Count fields must be polled periodically. 1 : Cell Count and Alternate Count fields will Generate an entry in the Count Rollover FIFO and rollover to 0 when their MSBs are set. Use this setting if the Count Rollover FIFO is being used.
0	COS_FIFO_enable	0 : No entries in the Change of State (COS) FIFO will be made by this connection. 1 : Changes in the Status field will be entered in the COS FIFO.

6.5.6 VC Table Address Settings

Settings to control the VC's addressing and address translation are contained in nine fields in its VC Table Record. Those fields are described in the Table 23. See the Datasheet [1] for further information.

Table 23 - VC Table Record Fields for Addressing Settings

Field	Size	Description
FieldB	12	The FieldB search key value that will be compared against the value extracted from incoming cells. Use the search key extraction parameters to determine this value.
VPI	12	The VPI associated with the connection. If the connection is a UNI then the first four bits should be set to zero.
VCI	16	The VCI associated with this connection. If this is a VPC connection then set this field to all zeroes.
Translated VPI	12	The VPI value that will be substituted on out-going cells if the global register bit XVPIVCI is set. If the global register bit XGFC is zero and the connection is a UNI (NNI = 1) then the first four bits of this field will not be substituted.
Translated VCI	16	The VCI value that will be substituted on out-going cells if the global register bit XVPIVCI is set.
Translated HEC	8	The HEC value that will be substituted on out-going cells if the global register bit XHEC is set.
Translated UDF	24	The UDF value that will be substituted on out-going cells if the global register bit XUDF is set.
Translated Pre/Po 1	32	The Translated Pre/Po 1 value that will be substituted on out-going cells if the global register bit XPREPO is set.
Translated Pre/Po 2	32	The Translated Pre/Po 1 value that will be substituted on out-going cells if the global register bit XPREPO is set.

Field	Size	Description
Bwds VCRA	16	The VC Record Address of the corresponding VC in the opposite-direction ATLAS-3200. If the OBCIF_Bwd_VCRA bit is logic 1, then this field will be inserted, along with the PHY ID and other routing information, in the prepended bytes of cells sent to the Backwards Cell Interface.

6.5.7 VC Table Policing Settings

Settings to control the VC's traffic rate policing are contained in ten fields in its VC Table Record. Those fields are described in Table 24. See the Data Sheet [1] for further information.

Table 24 - VC Record Fields for Policing Settings

Field	Size	Description
Policing Configuration	11	Control bits to configure policing settings. The bits that require coordinated settings to configure GCRA or GFR policing are described in Table 25. See the Data Sheet [1] for complete bit descriptions.
Maximum Frame Length	11	Guaranteed Frame Rate policing parameter. Set this according to GFR guidelines in Data Sheet [1] if this connection is transmitting packets.
GFR State	3	This is an internally maintained state variable that should initially be programmed to zero and not changed thereafter. . Write protection can be done using the write mask function when performing writes to the VC Table Record SRAM.
Policing Reserved	3	This should be set to 0's initially. Thereafter, it should not be overwritten. This can be done using the write mask function when performing writes to the VC Table Record SRAM.
Action 2	2	Selects the action taken on cells non-conforming with GCRA2. See Data Sheet [1] for action settings.
Inc 2	14	Parameter for GCRA2 based on desired traffic rate. See Data Sheet [1] for calculations to determine this value.

Field	Size	Description
Limit 2	14	Parameter for GCRA2 based on desired traffic rate. See Data Sheet [1] for calculations to determine this value.
Action 1	2	Selects the action taken on cells non-conforming with GCRA1. See Data Sheet [1] for action settings.
Inc 1	14	Parameter for GCRA1 based on desired traffic rate. See Data Sheet [1] for calculations to determine this value.
Limit 1	14	Parameter for GCRA1 based on desired traffic rate. See Data Sheet [1] for calculations to determine this value.

For a connection that is added over a PHY transmitting ATM cells, Generic Cell Rate Algorithm (GCRA) policing is used, and for a connection over a PHY transmitting packets Guaranteed Frame Rate (GFR) policing is used. Table 25 summarizes the Policing Configuration Field settings required to configure GCRA for GFR policing. The Policing Configuration Field bits not shown in Table 25 control independent policing options. See the Data Sheet [1] for further details.

Table 25 - GCRA and GFR Policing Configurations

Policing Mode	Policing Configuration Field Bits				
	MFL[10:0]	GFR_MCR_PPD	GFR	Action 1 [1:0]	Action 2 [1:0]
GCRA	X	X	0	any value	any value
GFR • partial packet discard • MFL test enabled	not all ones	1	1	11	01, 10, or 11
GFR • partial packet discard • MFL test disabled	0b1111111111				
GFR • actions on frame boundaries as per GFR standard • MFL test enabled	not all ones	0			
GFR • actions on frame boundaries as per GFR standard • MFL test disabled	0b1111111111				

X = don't care, this bit will not affect policing operation

6.5.8 VC Table OAM Settings

The settings to control OAM cell processing on a per-VC basis are discussed in Section 6.11 OAM Cell Processing. See that section for details on the OAM settings.

For a quick reference, the fields in the VC Table Records that are related to OAM cell processing are listed in Table 26.

Table 26 - VC Table Record Fields for OAM Settings

Field	Size	Description
OAM Configuration	23	Configuration bits. See Table 37 and Table 39 in Section 6.11 OAM Cell Processing for descriptions of the bit settings.
Segment Received Defect Type	8	See Table 37 in Section 6.11 OAM Cell Processing for descriptions of these fields
Segment Received Defect Location	128	
ETE Received Defect Type	8	
End-to-End Received Defect Location	128	

6.5.9 Performance Management Settings

To setup an F4 PM Session on a VPC that contains one or more VCCs, the VC Linkage Table Records of the VPC and VCCs must be configured as described in Section 6.5.3. To setup an F5 PM Session on a VCC or an F4 PM Session on an Independent VCC the VC Linkage Table Records have no special requirements.

See Section 6.11.3 PM Cell Processing for details on the performance management settings and algorithms for creating and linking to a PM Table Record.

6.5.10 Example Routines

6.5.10.1 Definitions

```
/************** Structures **********************/

typedef struct {
    UINT1    phyID
    BOOLEAN  pm2Active
    UINT1    pm2Addr
    BOOLEAN  pm1Active
    UINT1    pm1Addr
    BOOLEAN  vpcPointerActive
    UINT1    vpcPointer
} sVC_LINKAGE_TABLE

typedef struct {
```

```
    UINT1  phyID
    UINT4  fieldA
    UINT4  fieldB
    UINT2  VCI
    UINT4  VPI
} sROUTING_WORD
```

6.5.10.2 vcAddVCC

This routine adds a new Virtual Channel Connection (VCC). It optionally links the new VCC to an existing VPC.

Inputs:	addToVPC	: if TRUE then this VCC will link to the VPC VC Table Record at the address specified
	vcraVPC	: contains the VC Record Address of a VPC if addToVPC is TRUE
Outputs:	vcra	: VC Record Address where the VC Table Record was added.

Pseudocode:

```
VOID vcAddVCC(
    BOOLEAN  addToVPC
    UINT4    vcraVPC
    UINT4    *vcra
{
sVC_RECORD_TABLE    vcRecordTable
sVC_LINKAGE_TABLE   vcLinkageTable
BOOLEAN             validAddressFound
sROUTING_WORD       routingWord

/* write all the VC Table Record settings */
vcRecordTable.Configuration = /* desired setting */
vcRecordTable.fieldB = /* desired setting */
vcRecordTable.VCI = /* desired setting */
... /* set VC Table Record fields as desired */

/* optionally create a performance management session */
call routines to create one or two PM Table Records

/* write the VC Linkage Table settings */
```

```
if (addToVPC = TRUE) then  /* link to the VPC */
    vcLinkageTable.vpcPointerActive = TRUE
    vcLinkageTable.vpcPointer = vcraVPC
    ... /* set remaining Linkage fields as desired */
else /* VCC is independent so deactivate VPC Pointer */
    vcLinkageTable.vpcPointerActive = FALSE
    ... /* set remaining Linkage fields as desired */
end if

/* get a free VCRA address from the microprocessor maintained list of */
/* available addresses  */
if (addToVPC = TRUE) then
/* must get an address in a different bank than the VPC */

    /* the bank is determined by two LSBs of the vcra       */
    restrictedBank = (vcraVPC & 0x00000003)
    /* get a free address that is not in the restricted bank*/
    *vcra = getFreeVcraBankRestriction(restrictedBank)
else
   /* get any free address */
   *vcra = getFreeVCRA()
end if

/* write the VC Table Record and VC Linkage Table Records to Memory */
vcRecordTableWrite(vcra, &vcRecordTable)
sramWrite(vcra, SRAM_LINKAGE, vcLinkageTable)

/* enable the VC */
vcEnable(vcra)

/* insert this connection into the search tree*/
routingWord.phyID = vcLinkageTable.phyID
routingWord.VCI = vcRecordTable.VCI
routingWord.VPI = vcRecordTable.VPI
routingWord.fieldA =/*depends on search configuration and pre-postpend*/
routingWord.fieldB =/*depends on search configuration and pre-postpend*/
searchTreeInsert(vcra, routingWord)

return SUCCESS
}
```

6.5.10.3 vcAddVPC

This routine adds a new Virtual Path Connection (VPC). This function assignes the VPC's VC Table Record to an arbitrary memory bank. Therefore if any VCC's will supsequently be linked to this VPC, it is their responsibility to ensure

compatible addresses (VCCs in different memory bank than the VPC). This approach assumes that VPC will always be created first, before any VCCs which will be enclosed in it.

Inputs: (none)

Outputs: vcra : VC Record Address where the VC Table Record was added.

Pseudocode:

```
VOID vcAddVPC(
   UINT4    *vcra
{
sVC_RECORD_TABLE    vcRecordTable
sVC_LINKAGE_TABLE   vcLinkageTable
sROUTING_WORD       routingWord

/* write all the VC Table Record settings */
vcRecordTable.VCI = 0
vcRecordTable.Configuration = /* desired setting */
vcRecordTable.fieldB = /* desired setting */
... /* set remaining VC Table Record fields as desired */

/* optionally create a performance management session */
call routines to create one or two PM Table Records

/* write the VC Linkage Table Record settings */
vcLinkageTable.vpcPointerActive = FALSE
... /* set remaining Linkage fields as desired */

/* get a free VCRA address from the microprocessor maintained list  */
/* this assumes that any VCCs that will be linked to this have not  */
/* been created yet so there are no restrictions on which memory    */
/* bank can be used                                                 */
vcra = getFreeVCRA()

/* write the VC Table Record and VC Linkage Table Records to Memory */
vcRecordTableWrite(vcra, &vcRecordTable)
sramWrite(vcra, SRAM_LINKAGE, vcLinkageTable)

/* enable the VC */
vcEnable(vcra)
```

```
/* insert one entry into the search tree with a of VCI = 0. This will */
/* catch all VPC cells with VCI's 0-15                                 */
routingWord.phyID = vcLinkageTable.phyID
routingWord.VCI = 0
routingWord.VPI = vcRecordTable.VPI
routingWord.fieldA =/*depends on search configuration and pre-postpend*/
routingWord.fieldB =/*depends on search configuration and pre-postpend*/
searchTreeInsert(vcra, routingWord)

return SUCCESS
}
```

6.6 Virtual Connection Removal

This section is a guide to programming the virtual connection removal routines. When an independent VPC or a VCC is removed, its VC Table Record can be directly removed. To remove a VPC connection which contains explicit VCCs however, the VC Table entries for the VCCs should be before removed before removing the VPC's VC Table.

The sections below discuss the algorithm for removing a VC and provide example code.

6.6.1 Algorithm

1. Disable the VC by writing a logic 0 to the Active bit in the VC Table Record.
2. Modify the Search Tables to remove the VC Table Record and VC Linkage Table Record from the search tree. See the section 6.7 Virtual Connection Search Tree Modifying for details.
3. Clear the VC Linkage Table Record and VC Table Record.
4. If a performance management session was active for this VC, determine if it is still in use by other connections. The microprocessor should maintain a data structure containing a list of the free PM Table Records and for each active PM Table Record a list of which VCs are using it. If no other VCs are using the PM Table Record then mark it as a free table in the microprocessor's data structure.

6.6.2 Example Routines

6.6.2.1 vcRemove

This routine removes a Virtual Connection. The specific implementation of this routine will vary as the application specific methods for adding different connection types and the coordination of connections on multiple ATLAS-3200 devices will vary.

Inputs

device	Device to access

Outputs

vcra	Virtual Connection Record address

Pseudocode:

```
UINT1 vcRemove(
    UINT4 vcra
{
sVC_RECORD_TABLE vcRecordTable
sVC_LINKAGE_TABLE vcLinkageTable
UINT4               primaryKey
sSEC_SEARCH_KEY     secondaryKey

/* disable the VC */
vcDisable(vcra)
/* remove VC Table Record and VC Linkage Table Record */
searchTreeRemove(primaryKey, secondaryKey)

/* check if performance management session can be freed */
check data structure stored on microprocessor
if can be freed then
    update data structure
end if

/* clear all the VC Table Record settings */
vcRecordTable.Configuration = 0
vcRecordTable.bwdDirVCRA = 0
vcRecordTable.VCI = 0
  ...
vcRecordTable.OAMConfiguration = 0

/* clear all the VC Linkage Table Record settings */
vcLinkageTable.phyID = 0
  ...
vcLinkageTable.vpcPtr = 0

/* write the cleared VC Table Record and VC Linkage Table Record to  */
/* memory */
call routines to write to VC RAM and external SRAM

return SUCCESS
}
```

6.7 Virtual Connection Search Tree Modifying

This section is a guide to configuring the VC search engine and to programming the routines to traverse and modify the VC search tree structure.

The ATLAS-3200 uses a binary search tree to identify incoming cells and to determine the record in the VC Table with which they are associated. The binary search tree is stored in Search Tables which are located in external SRAM (see Figure 3). The search key is a configurable combination of the cell's prepend, postpend, header and PHYID that allows tailoring to the specific application. The search key configuration is controlled by the Search Engine Configuration Register (0x 10B) and is typically set once during the ATLAS-3200 initialization.

The ATLAS-3200 has the ability to navigate the search tree but the microprocessor has full responsibility for maintaining the structure of the search tree. When entries are added and removed from the search tree the microprocessor must navigate the search tree and make modifications to the Search Tables such that its integrity is maintained.

The following sections describe the search key settings, the search tree structure, and three search tree algorithms. The algorithms are for traversing a search tree, inserting a new record into a tree, and removing a record from a tree. Example pseudocode routines are provided in the last section to further demonstrate the search tree operations.

For a detailed example of a search tree scenario, see Appendix B: VC Binary Search Tree Example.

6.7.1 Search Key Settings

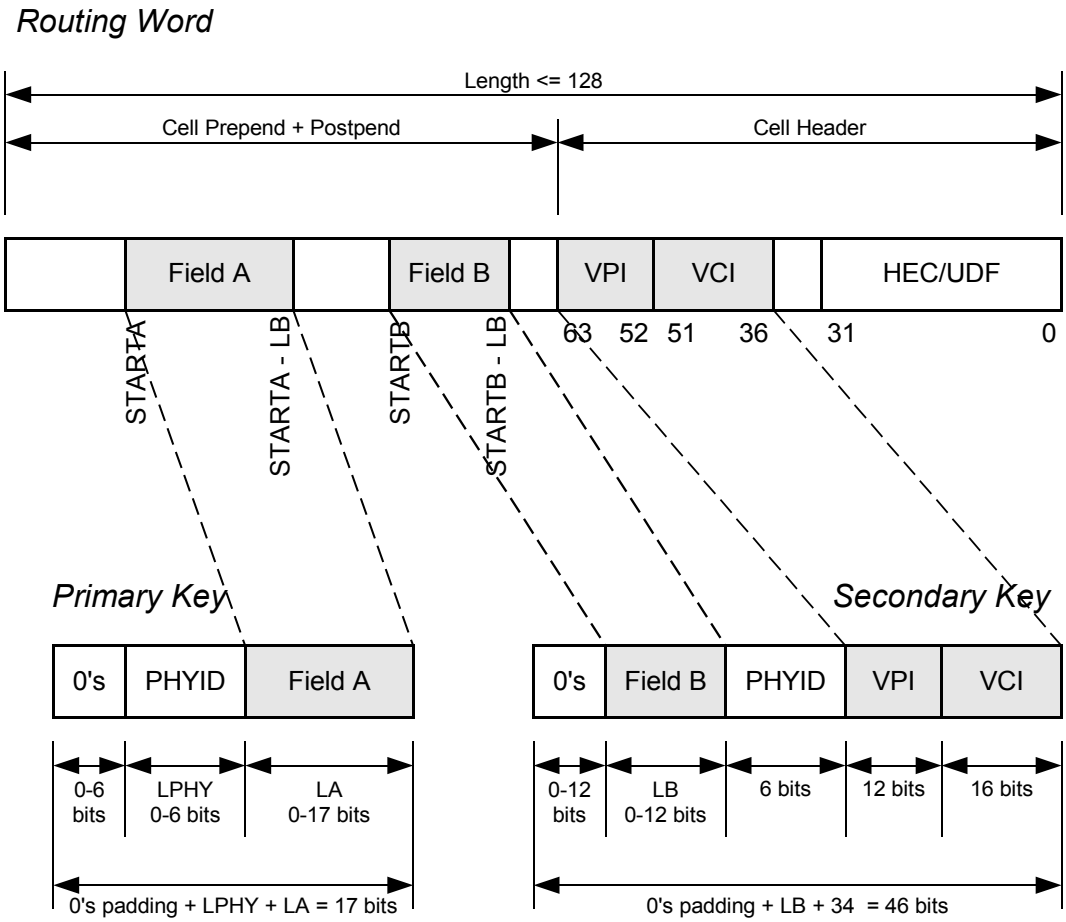
When a cell enters the ATLAS-3200, a Routing Word is constructed by concatenating its cell prepend, cell postpend, and cell header. A Primary and Secondary Search Key are then created by extracting selected portions of the Routing Word and the PHYID. The Search Engine Configuration Register (0x10B), described in Table 27, controls the Search Key. Figure 12 illustrates the Search Key construction. The Search Engine Configuration Register (0x10B) will typically be programmed during ATLAS-3200 initialization and may be configured as required for the application.

Table 27 - Search Engine Config. Register (0x10B) Fields

Field	Description
Search_From_IBCIF	0 : Cells from the IBCIF are assumed to carry a valid 16-bit VC Record Address and are not searched by the search engine. 1 : A search key is constructed from cells from the Input Backwards Cell Interface and the Search Tree is navigated to find the corresponding VC Record Address.
LPHY[2:0]	Number of PHY ID bits in the primary search key. The condition that LPHY + LA <= 17 must be met.
LA[4:0]	Length of the search key Field A field. The condition that LPHY + LA <= 17 must be met.
STARTA[6:0]	Location of the MSB of FieldA within the routing word
LB[3:0]	Length of the search key Field B field.
STARTB[6:0]	Location of the MSB of FieldB within the routing word

Note: *STARTA and STARTB specify the where the Most Significant Bits of the extracted fields will be, NOT the Least Significant Bits.*

Figure 12 - Search Key Construction



6.7.2 Search Tree Structure

The Search Tables are located in the lower partition of external SRAM as shown in Figure 3. At each Search Table address there is a Primary and Secondary Search entry, however, the Primary and Secondary Entries are treated independently. The Search Table structure is shown in Table 28 and the fields are described in Table 29.

Table 28 - Search Table Field Structure

Secondary Search Entry									Primary Search Entry
	Selector	Left Leaf		Left Branch	Right Leaf		Right Branch		Primary Search Pointer
[63:62]	[61:56]	[55]	[54]	[53:37]	[36]	[35]	[34:18]	[17]	[16:0]

Shaded cells = Reserved

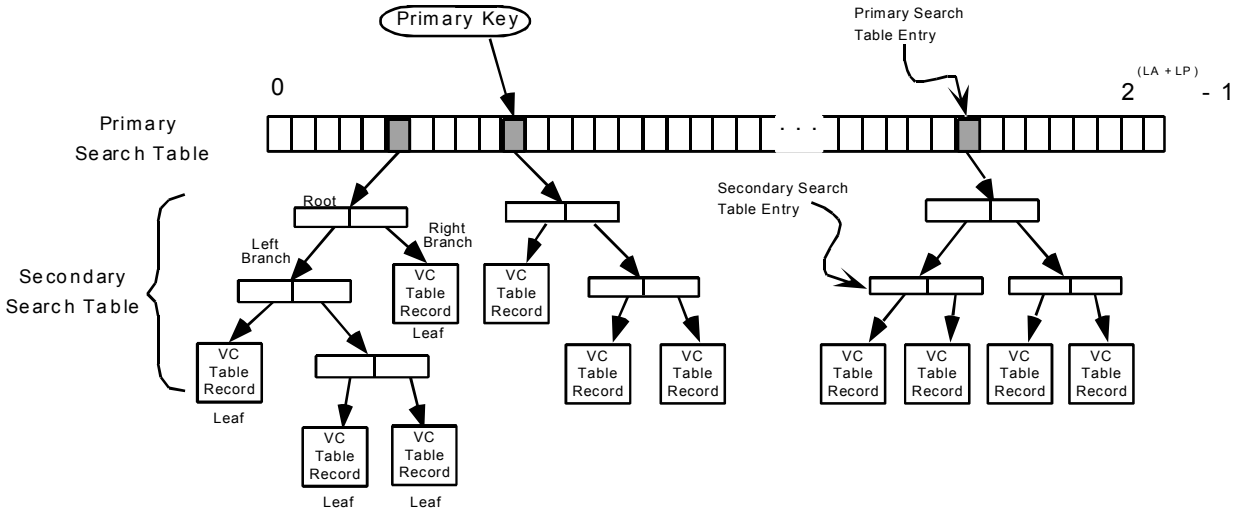
Table 29 - Search Table Field Descriptions

Field	Description
Primary Search Pointer[16:0]	Address of the Search Table entry that contains a root Secondary Search entry
Right Branch[16:0]	The pointer to the node accessed if the decision bit is a logic zero. If "Right Leaf" is a logic one, "Right Branch" contains the SA[15:0] address identifying the VC Table Record for the incoming cell. If "Right Leaf" is a logic zero, "Right Branch" contains the SA[15:0] value pointing to another Secondary Search Table entry.
Right Leaf	0 : Right Branch value points to another node in the binary tree. 1 : Right branch is a leaf and the binary search terminates if the decision bit is a logic zero.
Left Branch[16:0]	The 16-bit SRAM address pointing to the node accessed if the decision bit is a logic one. If "Left Leaf" is a logic one, "Left Branch" contains the SA[15:0] address identifying the VC Table Record for the incoming cell. If "Left Leaf" is a logic zero, "Left Branch" contains the SA[15:0] value pointing to another Secondary Search Table entry.
Left Leaf	0 : Left Branch value points to another node in the binary tree. 1 : Left Branch is a leaf and the binary search terminates if the decision bit is a logic one.

Field	Description
Selector[5:0]	Index to a bit in the Secondary Search Key that is the “Decision Bit” upon which the branching decision is based. A value of zero represents the LSB. If the Decision Bit is a logic one, the "Left Leaf" and "Left Branch" fields are subsequently used. Likewise, if the Decision Bit is a logic zero, the "Right Leaf" and "Right Branch" are subsequently used. Typically, the Selector value decreases monotonically with the depth of the tree, but other search sequences are supported by the flexibility of this bit. (i.e. typically, one starts from the most significant bit side and heads towards the least significant bit when selecting the bits to be used for branching decisions)

Each Primary Entry can have a binary tree composed of the Secondary Search Entries below it. An example search tree structure in shown in Figure 13. For a detailed search tree example see Appendix B: VC Binary Search Tree Example.

Figure 13 - Search Tree Structure



6.7.3 Algorithm for Finding a Record

Search tree traversing is used to find the VC Table Record and VC Linkage Table Record that a given ATM cell is associated with. This is automatically performed by the ATLAS-3200 for all arriving cells, and can also be performed by the microprocessor for search tree maintenance purposes. To find a search tree entry, the virtual connection's Prepend, Postpend, and Header fields must be known. The Search Tree can then be traversed to find the entry using the following algorithm:

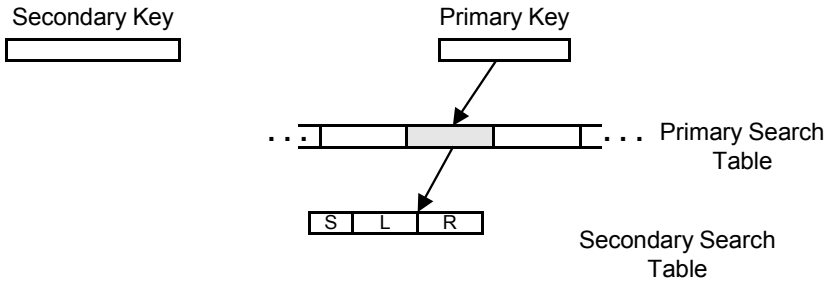
1. Construct a Routing Word from the VC's Prepend, Postpend, and Header.

2. Extract the Primary Search Key and a Secondary Search Key from the Routing Word and PHYID using the configuration specified in the Search Engine Configuration Register (0x10B).

Figure 14 - Search Tree Find, Step 2

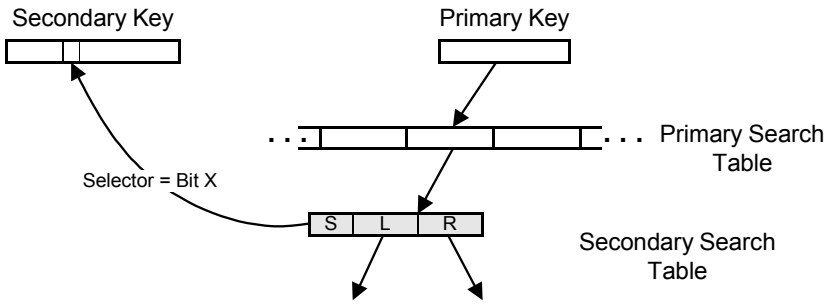
3. Read the Primary Search Entry at the address specified by the Primary Search Key. The value of the Primary Search Pointer field at this location addresses a Secondary Search Entry.

Figure 15 - Search Tree Find, Step 3



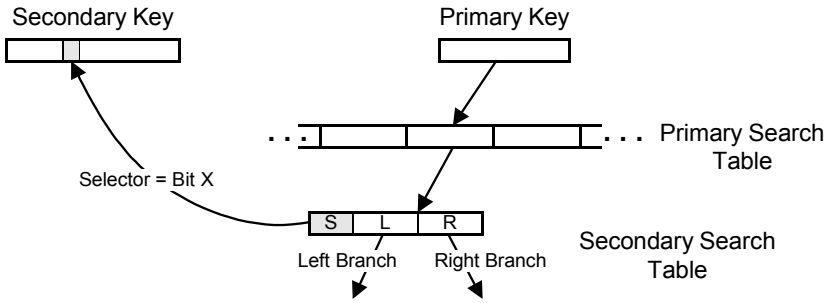
4. Read the Secondary Search Entry at the address specified by the Primary Search Entry. This Secondary Search entry is the root node of the binary search tree for this connection.

Figure 16 - Search Tree Find, Step 4



5. Look at the bit in the Secondary Search Key at the bit position identified by the Selector Field. (e.g. If Selector = 0 then look at the least significant bit, or if Selector = 3 then look at the fourth bit.) This bit serves as the Decision Bit to determine which direction to branch. If the Decision Bit is 0 then the right branch is taken, if the Decision Bit is 1 then the left branch is taken.

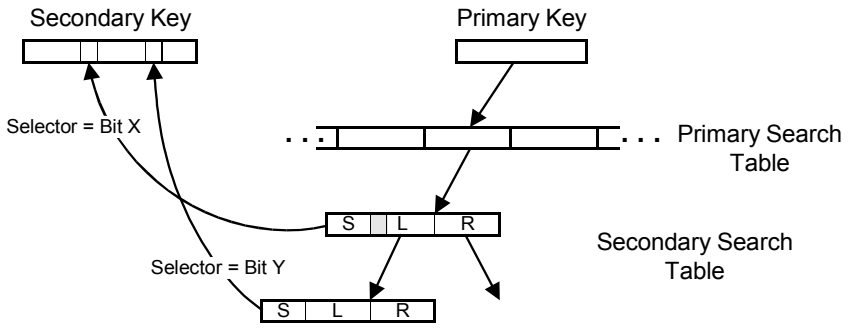
Figure 17 - Search Tree Find, Step 5



6. Look at the either the Right Leaf bit or the Left Leaf bit, as determined by the branch direction decision.

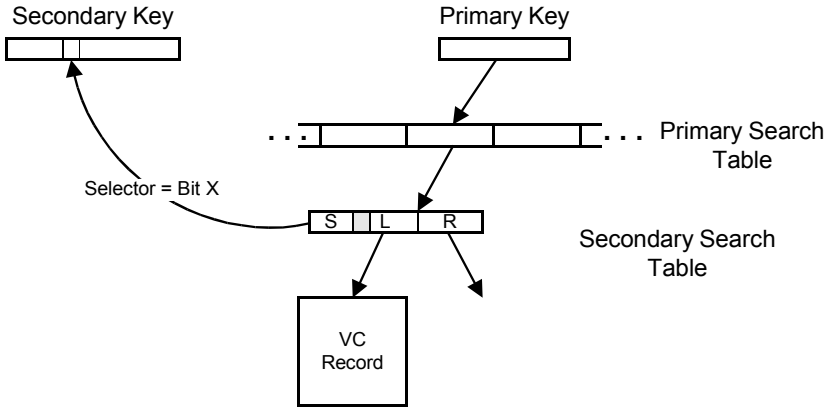
    - If the Leaf bit is 0 then this branch points to another Secondary Search Entry node in the tree and the search continues from the new node. Read the Secondary Search Entry at the address specified by the Branch field and LOOP back to Step 5.

Figure 18 - Search Tree Find, Step 6a



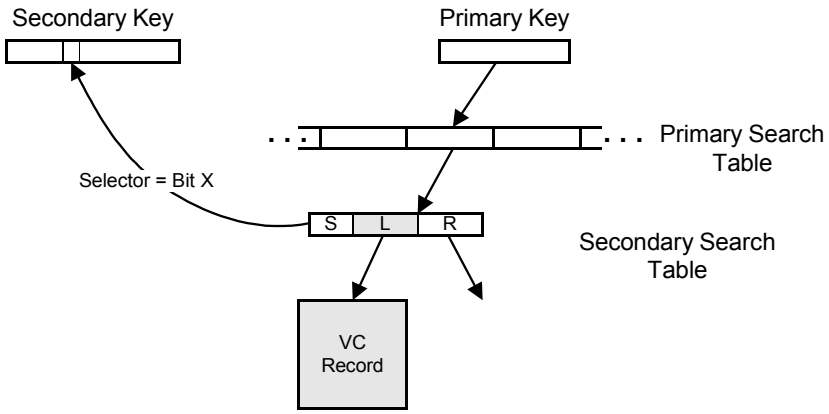
- If the Leaf bit is 1 then this is a leaf node and the Branch field contains the VC Record Address (VCRA) of a VC Table Record and VC Linkage Table Record. The search has terminated at a leaf, so CONTINUE to Step 7.

Figure 19 - Search Tree Find, Step 6b



7. Read the VC Table Record and VC Linkage Table Record at the address specified by the Branch field to determine leaf's Field B, VPI, VCI, and PHYID values. Compare these values to the Secondary Key used in the search to determine if there is a match. When the ATLAS-3200 is performing this search for an incoming cell, the cell is processed if a match is found. If a match is not found the cell is treated as an Unprovisioned Connection cell and will either be dropped or routed to the MCIF depending on the value of the BSDVCtoUP bit..

Figure 20 - Search Tree Find, Step 7



6.7.4 Algorithm for Inserting a Record

To insert a new VC Table Record into the search tree perform the following steps. This assumes that the new VC Table Record has been written to memory and its address is known.

1. Construct a Routing Word from the information in the VC Table Record, then construct the Primary and Secondary Search Keys using the Search Engine Configuration Register (0x10B) settings.

2. Traverse the search tree (using the replica VC Table structure) to determine the insertion point. The last pointer accessed in the search shall be the one modified, be it a Primary Search Table entry, left branch or right branch.

3. Locate a free Secondary Search Entry and write (see Section 7.2.2 Writing External SRAM Entries) initialization values to it. The only exception to this is when a single VC Table Record exists in a tree, in which case the solitary Secondary Search Table entry is modified.

4. Perform a single SRAM write (see Section 7.2.2 Writing External SRAM Entries) to incorporate the new Secondary Search Table entry in the existing tree structure. This step must be performed last to ensure a binary search in progress is not corrupted.

Five distinct insertions cases are possible based on the existing tree structure, and are detailed in the sections that follow. In the accompanying diagrams, the following key is used:

a, b, c: Pointers to Secondary Search Table records

w, x, y, z: Pointers to VC Table Records

k, m, n: Selector field contents

Shaded: Fields which have been modified in the process.

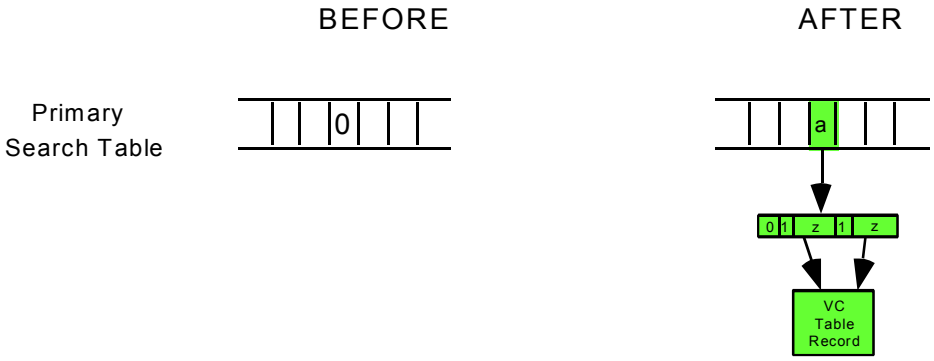
6.7.4.1 Case 1: Insertion into an Empty Tree

The binary tree is empty. There are no other connections that have the same Primary Search Key.

In this case, modify the null Primary Search Table pointer to point to a newly created Secondary Search Table entry. Because no bits within the Secondary Search Key are required, both the left and right branches of the Secondary Search Table entry point to the same VC Table Record. The selector is a 'don't care'.

NOTE: *The Secondary Search Table entry that is a root node CANNOT be at address 0. An entry of 0 in the Primary Search Pointer field is reserved to indicate an empty tree.*

Figure 21 - Search Tree Insertion into an Empty Tree

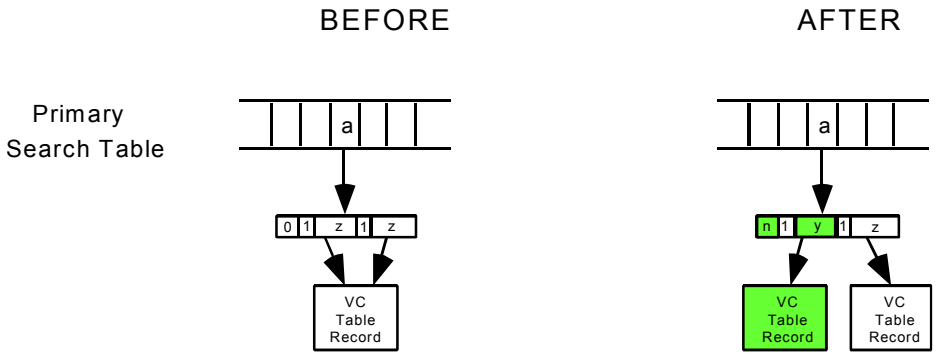


6.7.4.2 Case 2: Insertion into a Single Record Tree

The binary tree contains only a single VC Table Record.

Modify the selector field to index the most significant bit of the Secondary Search Key that differs between the new and existing connection. Modify the left or right branch, as appropriate, to point to the newly created VC Table Record.

Figure 22 - Search Tree Insertion into a Single Record Tree



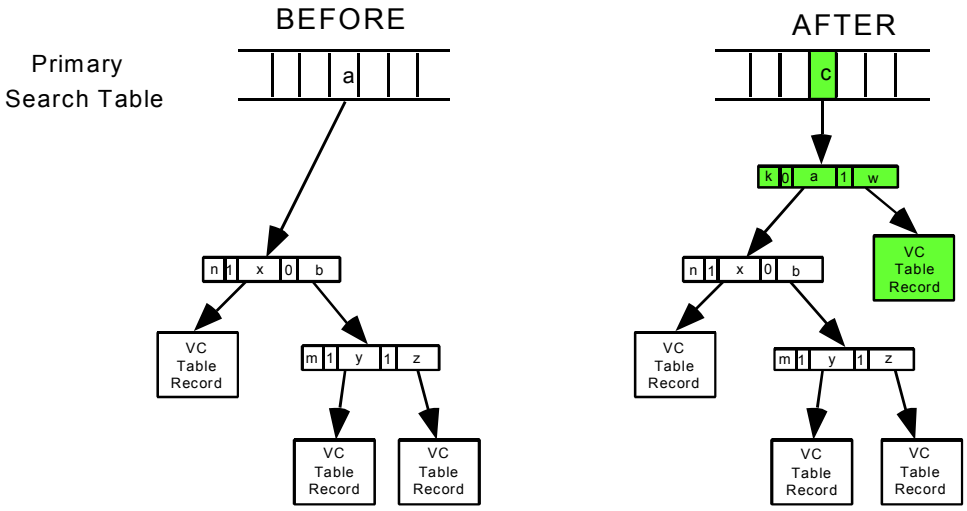
The diagram illustrates the case where the new VC has a one in the decision bit position and the existing VC has a zero in the same bit position. If the new VC has a zero in the decision bit position, modify the right branch instead.

6.7.4.3 Case 3: Insertion at Root of a Tree

The insertion point is at the root of the tree.

This occurs when the new decision bit index is greater any of the indices currently in the search tree. In this case, modify the Primary Search Table entry to point to the newly created Secondary Search Table entry. The New Secondary Search Table entry points to the new VC Table Record and the old tree root.

Figure 23 - Search Tree Insertion at the Root of a Tree

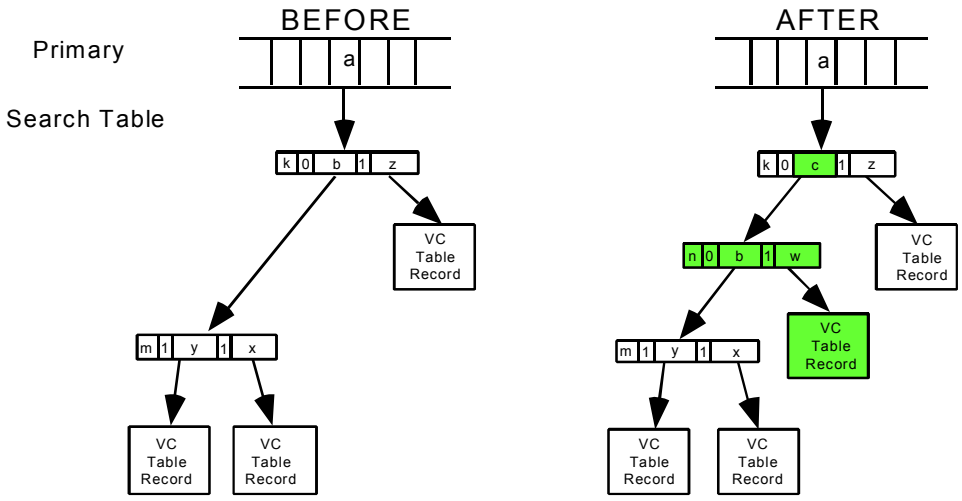


6.7.4.4 Case 4: Insertion at Middle of a Tree

The insertion point is in the middle of the binary tree. The new Secondary Search Table entry points to the new VC Table Record and an existing node in the tree.

Modify the parent of the existing node to point to the new Secondary Search Table entry in the final step of the insertion.

Figure 24 - Search Tree Insertion at Middle of a Tree

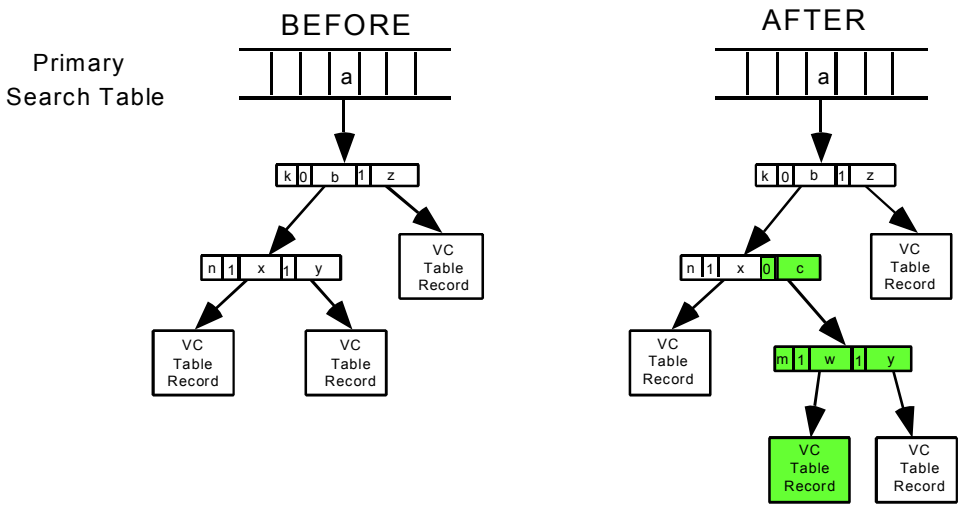


6.7.4.5 Case 5: Insertion at Leaf

The new Secondary Search entry is inserted at a leaf. The search for a candidate insertion point ends on a node which already points to a VC Table Record. The new Secondary Search entry points to the existing VC Table Record and the new VC Table Record.

Modify the existing Secondary Search Table entry to point to the new Secondary Search Table entry in the final step of the insertion.

Figure 25 - Search Tree Insertion at Leaf



6.7.5 Algorithm for Removing a Record

To remove a VC Table Record from the search tree perform the following steps.

The following are the microprocessor actions required to remove a connection:

1. Construct a Routing Word from the information in the VC Table Record, then construct the Primary and Secondary Search Keys using the Search Engine Configuration Register (0x10B) settings.

2. Traverse the search tree (using the replica VC Table structure) to locate the Secondary Search Table entry pointing to the connection's VC Table Record.

3. Perform an SRAM write to modify the parent node (be it the Primary Search entry or another Secondary Search entry) of the Secondary Search entry being removed to point to the node remaining after the connection removal. The only exception to this is when only two VC Table Records exist in a tree, in which case the solitary Secondary Search entry is modified. The VC is now considered unprovisioned and any cells belonging to the VC will be discarded.

4. Tag in software the removed Secondary Search entry as free.

5. Read the final statistics for the connection from the VC Table Record and tag in software the VC Table Record address as free. Also, clear the "Active" bit in the VC Table Record.

The search tree record removal process examples are not illustrated because the results are exactly the reverse of the search tree record insertion process. The five cases described in Section 6.7.4 Algorithm for Inserting a Record apply to removal if the Before and After diagrams are reversed.

6.7.6 Example Routines

6.7.6.1 Definitions

```
/************** Structures ***********************/

typedef struct {
    UINT1  phyID
    UINT4  fieldA
    UINT4  fieldB
    UINT2  VCI
    UINT4  VPI
} sROUTING_WORD

typedef struct {
    UINT1  selector
    UINT1  leftLeaf
    UINT4  leftBranch
    UINT1  rightLeaf
    UINT4  rightBranch
} sSEC_SEARCH_ENTRY

typedef UINT1 sSEC_SEARCH_KEY[6]  /* holds 46 bit secondary search key*/

typedef struct {
    UINT4    primaryKey       /* primary key used for the search */
    sSEC_SEARCH_KEY secondaryKey /* secondary key */
    BOOLEAN  matched          /* TRUE= search was successful */
    BOOLEAN  isRoot           /* TRUE= search terminated on root node */
    BOOLEAN  isEmptyTree      /* TRUE= no secondary entries for this */
                              /* primary key */
    BOOLEAN  isSingleBranch   /* TRUE = secondary tree has only one */
                              /* leaf */
    UINT4    leafAddr         /* address of secondary key found */
                              /* during the search */
    sSEC_SEARCH_KEY leafSecondaryKey /* secondary key of found record */
                              /* used for comparison to determine if */
                              /* search matched */
    UINT1    finalDirection   /* last search direction that the leaf */
                              /* was found on */
    UINT4    finalNodeAddr    /* address of final node searched */
```

```
    sSEC_SEARCH_ENTRY finalNode /*secondary search entry of final node*/
    UINT1    prevDirection    /* previous branch direction taken to */
                              /* reach the final node */
    UINT4    prevAddr         /* address of node that points to the */
                              /* final node  */
    UINT1    newSelector      /* most significant bit that is   */
                              /* different between the search  */
                              /* Secondary Key and the leaf's   */
                              /* Secondary Key */
} sSEARCH_RESULTS


/************** Global variables ****************/

/* Shadow copy of the Primary Search Table in the ATLAS-3200   */
/* external SRAM. */
UINT4 gPrimarySearchTable[]

/* Shadow copy of the Secondary Search Table in the ATLAS-3200 */
/* external SRAM. */
sSEC_SEARCH_ENTRY gSecondarySearchTable[]

/* Table of the Secondary Search Keys of all current VC Tables.*/
/* When a leaf has been reached in a search, the secondary key */
/* needs to be compared to determine if a match was found. This*/
/* table can be read instead of having to read the VC Record   */
/* Table and VC Linkage Table Record and recalculate the key.  */
sSEC_SEARCH_KEY gSecondarySearchKeys[]


/*************** Constant Definitions ***********/

/* Values for the decision bit chosen from Secondary Search Key*/
#define DECISION_BIT_LEFT                1
#define DECISION_BIT_RIGHT               0

/* Values for Branch fields in the Secondary Search Entry */
#define LEAF                             1
#define NOT_LEAF                         0

/* Values for the search direction bit in search results structure */
#define SEARCH_DIRECTION_LEFT            1
#define SEARCH_DIRECTION_RIGHT           0
```

6.7.6.2 searchTreeInsert

This routine inserts an entry into the search table for a new Virtual Connection. A search of the current search table is performed then the appropriate insertion routine is called based on the results of the search. It is assumed that the VC Table Record and VC Linkage have already been written to memory and their addresses are known.

Inputs: vcra : VC Record Address that the final Secondary Search Tree entry will point to.

routingWord : data structure containing the information about the connection that is required to construct the Primary and Secondary Search Keys,

Outputs: (none)

Pseudocode:

```
searchTreeInsert (
    UINT4               vcra
    sROUTING_WORD       routingWord
{
sSEARCH_RESULTS     *searchResults
sSEARCH_RESULTS     *insertionResults
UINT4               primaryKey
sSEC_SEARCH_KEY     secondaryKey

primaryKey = constructPrimaryKey(routingWord)
secondaryKey = constructSecondaryKey(routingWord)

searchTreeTraverse(primaryKey, secondaryKey, searchResults)

if (searchResults->matched = TRUE) then
    return ERROR_VC_EXISTS
end if

if (searchResults->isEmptyTree = TRUE)  /* case 1 */
    insertToEmptyTree(vcra)
else if (searchResults->isSingleBranch = TRUE)  /* case 2 */
    insertToSingleBranch(vcra, searchResults)
else if (searchResults->newSelector < searchResults->finalNode.selector)
    /* case 5 */
    insertToEndNode(vcra, searchResults)
else /* either case 3 or case 4 */
    /* In these cases the previous search traversed the tree past    */
```

```
    /* the point where the new leaf should be inserted based on the  */
    /* algorithm of decreasing selector values  So, perform a new    */
    /* search to find the correct insertion point                     */
    searchFindInsertionPoint(searchResults, insertionResults)
    if (insertResults->isRoot = TRUE) /* case 3 */
        insertToRoot(vcra, insertionResults)
    else /* case 4 by default */
        insertToMiddleBranch(vcra, insertionResults)
    end if
end if

return
}
```

6.7.6.3 searchTreeTraverse

This routine traverses the search tree using the specified primary and secondary search keys until a leaf is reached. The results of the tree traversal are stored in the search results structure. The results are used by other routines to determine how to modify the tree to add or remove a leaf.

Inputs:	primaryKey	: key used to locate Primary Search Entry
	secondaryKey	: key used to traverse the binary search tree composed of Secondary Search Entries
Outputs:	searchResults	: results from the search tree traversal. Contains information about the nodes surrounding the location where the search terminated.

Pseudocode:

```
searchTreeTraverse(
    UINT4                primaryKey,
    sSEC_SEARCH_KEY      secondaryKey,
    sSEARCH_TREE_RESULTS *searchResults)
{
BOOLEAN                    foundLeaf
UINT1                      decisionBit
UINT1                      currentDirection
UINT4                      previousNodeAddr
sSEC_SEARCH_ENTRY          curentNode
UINT4                      currentNodeAddr
UINT4                      nextNodeAddr
```

```
searchResultsInitialize(searchResults) /*initialize result values*/

/* get address of the root node in the secondary search tree */
currentNodeAddr = gPrimarySearchTable[primaryKey]

if (currentNodeAddr = 0) then  /* case 1 - the search tree is empty */
    searchResults->isEmptyTree = TRUE
    return
end if

/* initialize while loop variables */
previousNodeAddr = currentNodeAddr
foundLeaf = FALSE

while (foundLeaf = FALSE)
    currentNode = gSecondarySearchTable[currentNodeAddr]
    currentDirection = getDecisionBit(secondaryKey,
                                      currentNode.selector)
    if (currentDirection = DECISION_BIT_LEFT) then /* go left */
        if (currentNode.leftLeaf = LEAF) then
            foundLeaf = TRUE
        end if
        nextNodeAddr = currentNode.leftBranch
    else /* go right */
        if currentNode.rightLeaf = LEAF
            foundLeaf = TRUE
        end if
        nextNodeAddr = currentNode.rightBranch
    end if

    if (foundLeaf = FALSE)  /* continue search */
        searchResults->previousDirection = currentDirection
        previousNodeAddr = currentNodeAddr
        currentNodeAddr = nextNodeAddr
    end if
end while

/* a leaf has been reached so save the results of the search */
searchResults->previousAddr = previousNodeAddr
searchResults->finalDirection = currentDirection
searchResults->finalNodeAddr = currentNodeAddr
searchResults->finalNode = gSecondarySearchTable[currentNodeAddr]
searchResults->leafAddr = nextNodeAddr
searchResults->leafSecondaryKey = gSecondarySearchKeys[nextNodeAddr]
searchResults->secondary = secondaryKey
searchResults->primary = primaryKey
```

```
/* test for case 3, at the root of the tree */
if (previousNodeAddr = currentNodeAddr) then
    searchResults->isRoot = TRUE
end if

/* test for case 2, only a single branch in the tree*/
if ( (searchResults->finalNode.leftBranch =
          searchResults->finalNode.rightBranch)
      && (searchResults->finalNode.leftLeaf = LEAF)
      && (searchResults->finalNode.rightLeaf = LEAF)
      && (searchResults->isRoot) = TRUE) ) then
    searchResults->isSingleBranch = TRUE
end if

/* if the secondary key of the new VC matches the secondaryKey   */
/* of the the VC Record leaf that the search terminated on, then */
/* the VC already exists.*/
if (secondaryKey = searchResults->leafSecondaryKey) then
    searchResults->matched = TRUE
else /* find the most significant bit that differs between the keys */
    searchResults->matched = FALSE
    searchResults->newSelector = findMostSigDifferentBit( secondaryKey,
                                        searchResults->leafSecondaryKey)
end if

return
}
```

6.7.6.4 searchTreeFindInsertionPoint

This routine traverses the search tree to find the appropriate location to insert a new leaf. The location of the leaf is chosen so the selector values of the nodes will follow decrementing values down the search tree levels.

In insertion Case 3 and Case 4, the searchTreeTraverse routine will traverse the search tree past the location where the new leaf should be inserted. This routine will be called in those cases, and the results of a preceding searchTreeTraverse are used as the basis to start this search.

Inputs: searchResults : results from a previous search tree traversal performed by searchTreeTraverse routine. Provides the Search Tree information reuired to start the new search.

Outputs: insertResults : results from the insertion point Search Tree traversal. Contains information about the nodes surrounding the location where the new leaf should be inserted.

Pseudocode:

```
searchTreeFindInsertionPoint(
    sSEARCH_RESULTS     *searchResults
    sSEARCH_RESULTS     *insertResults)
{
BOOLEAN             foundInsertionPoint
UINT1               currentDirection
UINT4               previousNodeAddr
UINT4               currentNodeAddr
UINT4               nextNodeAddr
sSEC_SEARCH_ENTRY   curentNode

searchResultsInitialize(insertResults) /*initialize result values*/

/* copy the search keys from the previous search */
insertResults->secondaryKey = searchResults->secondaryKey
insertResults->primaryKey = searchResults->primaryKey

/* start search at top of the tree */
currentNodeAddr = gPrimarySearchTable[insertResults->primaryKey]

/* initialize while loop variables */
previousNodeAddr = currentNodeAddr
foundInsertionPoint = FALSE

while (foundInsertionPoint = FALSE)
    currentNode = gSecondarySearchTable[currentNodeAddr]

    /* if the selctor value of the current node is less than the    */
    /* new selector value determined by the search results then     */
    /* the leaf should be inserted here so that the selector values */
    /* follow decrementing values down the search tree levels       */
    if (currentNode.selector < searchResults.newSelector)
        foundInsertionPoint = TRUE
    else
        /* continue the selector value comparison at the node at the */
        /* next level in the tree */
        currentDirection = searchGetDecisionBit(
                                        insertResults->secondaryKey,
```

```
                                                currentNode.selector)
            if (currentDirection = DECISION_BIT_LEFT) then /* go left */
                nextNodeAddr = currentNode.leftBranch
            else /* go right */
                nextNodeAddr = currentNode.rightBranch
            end if

            insertResults->previousDirection = currentDirection
            previousNodeAddr = currentNodeAddr
            currentNodeAddr = nextNodeAddr
        end if
    end while

    /* the insertion point has been reached so save the results of this */
    /* search */
    insertResults->finalDirection = currentDirection
    insertResults->newSelector = searchResults->newSelector
    insertResults->finalNodeAddr = currentNodeAddr
    insertResults->finalNode = currentNode

    /* check if the search ended on the root node, (Case 3) */
    if (insertResults->currentNodeAddr=insertResults->previousNodeAddr) then
        insertResults->isRoot = TRUE
    else
        insertResults->isRoot = FALSE
    end if

    return
    }
```

6.7.6.5 insertToEmptyTree - (Case 1)

This routine inserts an entry into an empty search tree. It will be called by searchTreeInsert when insertion Case 1, as shown in Figure 21, occurs.

Inputs:

vcra	: VC Record Address that the final Secondary Search Tree entry will point to.
secondaryKey	: key associated with this vcra that will be written to one of the Search Key Tables

Outputs: (none)

Pseudocode:

```
insertToEmptyTree (
    UINT4               vcra
    sSEC_SEARCH_KEY     secondaryKey)
{
UINT4               rootNodeAddr
sSEC_SEARCH_ENTRY   rootNode

/* get an address for the new secondary search entry              */
/* Since this is a root node, be aware that it cannot be assigned */
/* address 0 since this is reserved for the empty tree condition. */
/* The function getFreeSecondaryAddress() is defined to never     */
/* return address 0 to prevent this.                              */
rootNodeAddr = getFreeSecondaryAddress()

/* initialize new root node */
/* since only node and one leaf in the tree set both branches */
/* to point to the leaf     */
rootNode.leftLeaf = LEAF
rootNode.leftBranch = vcra
rootNode.rightLeaf = LEAF
rootNode.rightBranch = vcra
/* now there is only one leaf so the selector bit has no effect     */
/* It will be set once another leaf is added (Case 2)               */
rootNode.selector = 0

/* write the root node to external SRAM and to shadow copy */
searchTreeWriteSecondaryEntry(rootNodeAddr, rootNode)
searchTreeWriteShadowSecondaryEntry(rootNodeAddr, rootNode)
/* save this search key to the microprocessor's table      */
gSecondarySearchKeys[vcra] = secondaryKey

/* write the primary pointer to shadow copy and to the external SRAM  */
/* This write will update the search tree to include the root node    */
/* and its leaf in a single atomic operation */
searchWritePrimaryEntry(primaryKey, rootNodeAddr)
searchWriteShadowPrimaryEntry(primaryKey, rootNodeAddr)

return
}
```

6.7.6.6 insertToSingleBranch - (Case 2)

This routine inserts a leaf into a search tree that only contains a single node and a single leaf. It will be called by searchTreeInsert when insertion Case 2, as shown in Figure 22, occurs.

Inputs:

vcra	: VC Record Address that the final Secondary Search Tree entry will point to.
searchResults	: results from a search tree traversal. Contains information about the nodes surrounding the location where the leaf will be inserted.
secondaryKey	: key associated with this vcra that will be written to one of the Search Key Tables

Outputs: (none)

Pseudocode:

```
insertToSingleBranch (
    UINT4               vcra
    sSEARCH_RESULTS     *searchResults
    sSEC_SEARCH_KEY     secondaryKey)
{
UINT1               decisionBit
sSEC_SEARCH_ENTRY   modifiedRootNode
UINT4               modifiedRootNodeAddr

modifiedRootNode = searchResults->finalNode
modifiedRootNodeAddr = searchResults->finalNodeAddr

/* initialize a modified version of the current root node so that   */
/* one of the branches now points to the new leaf. the other branch */
/* will still be pointing to the existing leaf */
modifiedRootNode.selector = searchResults->newSelector
decisionBit = searchGetDecisionBit(searchResults->secondaryKey,
                                   modifiedRootNode.selector)
if (decisionBit = DECISION_BIT_LEFT) then /* put new leaf on the left */
    modifiedRootNode.leftBranch = vcra
else /* put new leaf on the right */
    modifiedRootNode.rightBranch = vcra
end if

/* overwrite current end node in the external SRAM and shadow copy */
/* This write will update the search tree to include the new leaf     */
/* in a single atomic operation                                       */
searchTreeWriteSecondaryEntry(modifiedRootNodeAddr, modifiedRootNode)
searchTreeWriteShadowSecondaryEntry(modifiedRootNodeAddr,
                                    modifiedRootNode)
/* save this search key to the microprocessor's table      */
gSecondarySearchKeys[vcra] = secondaryKey
```

```
return
}
```

6.7.6.7 insertToRoot - (Case 3)

This routine inserts a new root at the top of a search tree and places the new leaf under it. It will be called by searchTreeInsert when insertion Case 3, as shown in Figure 23, occurs.

Inputs:

vcra	: VC Record Address that the final Secondary Search Tree entry will point to.
searchResults	: results from a search tree traversal. Contains information about the nodes surrounding the location where the leaf will be inserted.
secondaryKey	: key associated with this vcra that will be written to one of the Search Key Tables

Outputs: (none)

Pseudocode:

```
insertToRoot (
    UINT4               vcra
    sSEARCH_RESULTS     *insertResults
    sSEC_SEARCH_KEY     secondaryKey)
{
UINT1               decisionBit
UINT4               newRootNodeAddr
sSEC_SEARCH_ENTRY   newRootNode


/* initialize the new root node to point to the new leaf and to the */
/* previous root node                                               */
newRootNode.selector = insertResults->newSelector
decisionBit = searchGetDecisionBit(insertResults->secondaryKey,
                                   newRootNode.selector)
if (decisionBit = DECISION_BIT_LEFT) then /* put leaf on left */
    newRootNode.leftLeaf = LEAF
    newRootNode.leftBranch = vcra
    newRootNode.rightLeaf = NOT_LEAF
    newRootNode.rightBranch = insertResults->finalNodeAddress
```

```
else /* put leaf on right */
    newRootNode.leftLeaf = NOT_LEAF
    newRootNode.leftBranch = insertResults->finalNodeAddress
    newRootNode.rightLeaf = LEAF
    newRootNode.rightBranch = vcra
end if

/* write the new root node to the external SRAM and shadow copy   */
searchTreeWriteSecondaryEntry(newRootNodeAddr, newRootNode)
searchTreeWriteShadowSecondaryEntry(newRootNodeAddr, newRootNode)
/* save this search key to the microprocessor's table      */
gSecondarySearchKeys[vcra] = secondaryKey


/* write the primary pointer to the shadow copy and the external SRAM */
/* This write will update the search tree to include the new root     */
/* node and its leaf in a single atomic operation */
searchWritePrimaryEntry(insertResults->primaryKey, newRootNodeAddr)

return
}
```

6.7.6.8 insertToMiddleBranch - (Case 4)

This routine inserts a leaf into the middle of a search tree. It will be called by searchTreeInsert when insertion Case 4, as shown in Figure 24 occurs.

Inputs:

vcra	: VC Record Address that the final Secondary Search Tree entry will point to.
searchResults	: results from a search tree traversal. Contains information about the nodes surrounding the location where the leaf will be inserted.
secondaryKey	: key associated with this vcra that will be written to one of the Search Key Tables

Outputs: (none)

Pseudocode:

```
insertToMiddleBranch (
    UINT4               vcra
    sSEARCH_RESULTS     *insertResults
    sSEC_SEARCH_KEY     secondaryKey)
{
```

```
UINT1                decisionBit
sSEC_SEARCH_ENTRY    newNode
UINT4                newNodeAddr
sSEC_SEARCH_ENTRY    modifiedNode
UINT4                modifiedNodeAddr

newNodeAddr = getFreeSecondaryAddress()

/* initialize the new node to point to the new leaf and to the */
/* next node below it                                          */
newNode.selector = insertResults.newSelector
decisionBit = searchGetDecisionBit(insertResults->secondaryKey,
                                   newNode.selector)
if (decisionBit = DECISION_BIT_LEFT) then /* put leaf on left */
    newRootNode.leftLeaf = LEAF
    newRootNode.leftBranch = vcra
    newRootNode.rightLeaf = NOT_LEAF
    newRootNode.rightBranch = insertResults->finalNodeAddress
else /* put leaf on right */
    newRootNode.leftLeaf = NOT_LEAF
    newRootNode.leftBranch = insertResults->finalNodeAddress
    newRootNode.rightLeaf = LEAF
    newRootNode.rightBranch = vcra
end if

/* initialize the previous node in the table to point to the new */
/* node */
modifiedNode = insertResults->finalNode
modifiedNodeAddr = insertresults->finalNodeAddr

if (insertResults->previousDirection = DIRECTION_LEFT) then
    /* Left path from the previous node was taken, so insert the */
    /* new node below the previous node's left branch */
    modifiedNode.leftBranch = newNodeAddr
    modifiedNode.leftLeaf = NOT_LEAF
else
    /* Right path from the previous node was taken, so insert the */
    /* new node below the previous node's left branch */
    modifiedNode.rightBranch = newNodeAddr
    modifiedNode.rightLeaf = NOT_LEAF
end if

/* first write the new node to the shadow copy and external SRAM     */
/* this will not affect the current tree structure                   */
searchTreeWriteSecondaryEntry(newNodeAddr, newNode)
searchTreeWriteShadowSecondaryEntry(newNodeAddr, newNode)
/* save this search key to the microprocessor's table      */
```

```
gSecondarySearchKeys[vcra] = secondaryKey


/* overwrite the modified node in the shadow copy and external SRAM  */
/* This write will update the search tree to include the new node    */
/* in a single atomic operation                                      */
searchTreeWriteSecondaryEntry(modifiedNodeAddr, modifiedNodeAddr)
searchTreeWriteShadowSecondaryEntry(modifiedNodeAddr, modifiedNodeAddr)
/* save this search key to the microprocessor's table      */
gSecondarySearchKeys[vcra] = secondaryKey

return
}
```

6.7.6.9 insertToEndNode - (Case 5)

This routine inserts a leaf at the end of a search tree . It will be called by searchTreeInsert when insertion Case 5, as shown in Figure 25, occurs.

Inputs:

vcra	: VC Record Address that the final Secondary Search Tree entry will point to.
searchResults	: results from a search tree traversal. Contains information about the nodes surrounding the location where the leaf will be inserted.
secondaryKey	: key associated with this vcra that will be written to one of the Search Key Tables

Outputs: (none)

Pseudocode:

```
insertToEndNode (
    UINT4               vcra
    sSEARCH_RESULTS     *searchResults
    sSEC_SEARCH_KEY     secondaryKey)
{
UINT1               decisionBit
sSEC_SEARCH_ENTRY   newEndNode
UINT4               newEndNodeAddr
sSEC_SEARCH_ENTRY   modifiedNode
UINT4               modifiedNodeAddr

newEndNodeAddr = getFreeSecondaryAddress()
```

```
modifiedNodeAddr = searchResults->finalNodeAddr

/* initialize a modified version of the current end node so that   */
/* the branch that pointed to the final leaf now points to the new */
/* end node */
modifiedNode = searchResults->finalNode
if (searchResults->finalDirection = SEARCH_DIRECTION_LEFT) then
    /* put new node on the left */
    modifiedNode.leftLeaf = NOT_LEAF
    modifiedNode.leftBranch = newEndNodeAddr
else /* put new node on the right */
    modifiedNode.rightLeaf = NOT_LEAF
    modifiedNode.rightBranch = newEndNodeAddr
end if

/* initialize the new end node so that it points to the new leaf */
/* and to the leaf that the current end node points to           */
newEndNode.selector = searchResults->newSelector
newEndNode.leftLeaf = LEAF
newEndNode.rightLeaf = LEAF
decisionBit = searchGetDecisionBit(searchResults->secondaryKey,
                                   newEndNode.selector)
if (decisionBit = DECISION_BIT_LEFT) then
    newEndNode.leftBranch = vcra
    newEndNode.rightBranch = searchResults->leafAddress
else /* DECISION_BIT_RIGHT */
    newEndNode.leftBranch = searchResults->leafAddress
    newEndNode.rightBranch = vcra
end if

/* first write the new end node in the shadow copy and external SRAM */
/* this will not affect the current tree structure                   */
searchTreeWriteSecondaryEntry(newEndNodeAddr, newEndNode)
searchTreeWriteShadowSecondaryEntry(newEndNodeAddr, newEndNode)
/* save this search key to the microprocessor's table      */
gSecondarySearchKeys[vcra] = secondaryKey


/* overwrite current end node in the shadow copy and external SRAM    */
/* This write will update the search tree to include the new end node */
/* in a single atomic operation                                       */
searchTreeWriteSecondaryEntry(modifiedEndNodeAddr, modifiedNode)
searchTreeWriteShadowSecondaryEntry(modifiedEndNodeAddr, modifiedNode)
/* save this search key to the microprocessor's table      */
gSecondarySearchKeys[vcra] = secondaryKey
```

```
return
}
```

6.7.6.10 searchTreeRemove

This routine removes the entry from the search table associated with a given VC Table Record. The Search Table is traversed to find the leaf entry, then the appropriate removal routine is called based on the tree structure.

Inputs: routingWord : data structure containing all the VC's information that is required to construct the Primary and Secondary Search Keys

Outputs: (none)

Pseudocode:

```
UINT1 searchTreeRemove (
    sROUTING_WORD       routingWord
{

sSEARCH_RESULTS     *searchResults

primaryKey = constructPrimaryKey(routingWord)
secondaryKey = constructSecondaryKey(routingWord)

searchTreeTraverse(primaryKey, secondaryKey, searchResults)

if (searchResults->matched = FALSE) then
    return ERROR_VC_NOT_EXIST
end if

if (searchresults->isRoot = FALSE) then
    /* remove a node that is not the root node, (Case 4 and Case 5) */
    removeEndNode(searchResults)
else /* the root node is involved, (Case 1, Case 2, and Case 3) */
    if (searchResults->singleBranch = TRUE ) then
        /* Case 1 */
        removeSingleBranch(searchResults)
    else if (searchResults->finalNode.leftLeaf = LEAF)
            and (searchResults->finalNode.rightLeaf = LEAF) then
        /* Case 2 */
        removeDoubleBranch(searchResults)
    else /* Case 3 */
```

```
            removeRootNode(searchResults)
        end if
    end if

    return SUCCESS
}
```

6.7.6.11 removeSingleBranch - (Case 1)

This routine removes the leaf from a table that contains only a single node and a single leaf. It will be called by searchTreeRemove when the mirror of insertion Case 1, as shown in Figure 21, occurs.

Inputs: searchResults : results from a search tree traversal. Contains information about the nodes surrounding the the leaf that will be removed.

Outputs: (none)

Pseudocode:

```
removeSingleBranch (
    sSEARCH_RESULTS     *searchResults)
{

/* set the primary entry to all zeroes to indicate the tree */
/* for this primary key is empty */
searchWritePrimaryEntry(insertResults->primaryKey, 0)

/* mark this secondary search address as available for use */
releaseSecondaryAddress(searchResults->finalNodeAddr)

return
}
```

6.7.6.12 removeDoubleBranch - (Case 2)

This routine removes a leaf from a table that contains a single node and two leaves. It will be called by searchTreeRemove when the mirror of insertion Case 2, as shown in Figure 22, occurs.

Inputs: searchResults : results from a search tree traversal. Contains information about the nodes surrounding the the leaf that will be removed.

Outputs: (none)

Pseudocode:

```
removeDoubleBranch (
    sSEARCH_RESULTS     *searchResults)
{

UINT1               decisionBit
sSEC_SEARCH_ENTRY   modifiedRootNode
UINT4               modifiedRootNodeAddr

/* initialize a modified version of the current root node so that   */
/* both of the branches now points to single remaining leaf.        */

modifiedRootNode = searchResults->finalNode
modifiedRootNodeAddr = searchResults->finalNodeAddr

/* now there will be only one leaf with both branches pointing to   */
/* it so the selector bit has no effect */
modifiedRootNode.selector = 0

if (searchResults->finalDirection = DIRECTION_LEFT) then
    /* remove left leaf  - left now points to same as right */
    modifiedRootNode.leftBranch = modifiedRootNode.rightBranch
else
    /* remove right leaf  - right now points to same as left */
    modifiedRootNode.rightBranch = modifiedRootNode.leftBranch
end if

/* overwrite the modified root node in the shadow copy and external */
/* SRAM. This write will update the search tree to remove the old   */
/* leaf in a single atomic operation                                */
searchTreeWriteSecondaryEntry(modifiedNodeAddr, modifiedNodeAddr)
searchTreeWriteShadowSecondaryEntry(modifiedNodeAddr, modifiedNodeAddr)
/* save this search key to the microprocessor's table      */
gSecondarySearchKeys[vcra] = secondaryKey

return
}
```

6.7.6.13 removeRootNode - (Case 3)

This routine removes a leaf from a root node that has the leaf on one branch and a node on its other branch. It will be called by searchTreeRemove when the mirror of insertion Case 3, as shown in Figure 23, occurs.

Inputs: searchResults : results from a search tree traversal. Contains information about the nodes surrounding the the leaf that will be removed.

Outputs: (none)

Pseudocode:

```
removeRootNode (
    sSEARCH_RESULTS     *searchResults)
{

UINT1               decisionBit
UINT4               newRootNodeAddr

/* make the second entry in the tree the new root node             */
/* A root node is assigned here, so be aware that it cannot be     */
/* have address 0 since this is reserved for empty tree condition. */
/* However, since address 0 is never assigned by the               */
/* getFreeSecondaryAddress() function this will not happen here    */
if searchResults->finalDirection = DIRECTION_LEFT then
    newRootNodeAddr = searchResults->finalNode.rightBranch
else
    newRootNodeAddr = searchResults->finalNode.leftBranch
end if

/* overwrite primary entry in the shadow copy and external SRAM  */
/* This write will update the search tree to remove the old root */
/* in a single atomic operation                                  */
searchWritePrimaryEntry(insertResults->primaryKey, newRootNodeAddr)

/* mark the secondary search address of the old root node as  */
/* available for use */
releaseSecondaryAddress(searchResults->finalNodeAddr)

return
}
```

6.7.6.14 removeEndNode - (Case 4 and Case 5)

This routine removes a leaf from any node that is not the root node. It will be called by searchTreeRemove when the mirror of insertion Case 3 or Case 4, as shown in Figure 24 and Figure 25, occurs.

Inputs: searchResults : results from a search tree traversal. Contains information about the nodes surrounding the the leaf that will be removed.

Outputs: (none)

Pseudocode:

```
searchTreeRemoveEndNode (
    sSEARCH_RESULTS     *searchResults)
{

UINT1               decisionBit
sSEC_SEARCH_ENTRY   modifiedPrevNode
UINT4               modifiedPrevNodeAddr
sSEC_SEARCH_ENTRY   removedNode
UINT4               removedNodeAddr

modifiedGrandParentNodeAddr = searchResults->prevAddr
modifiedGrandParentNode =
                  gSecondarySearchTable[modifiedGrandParentNodeAddr]

removedParentNode = searchResults->finalNode
removedParentNodeAddr = searchResults->finalNodeAddr

/* the parent node of the removed leaf has one branch pointing   */
/* to the leaf that will be removed and another branch pointing  */
/* to the rest of the search tree below it. The parent node will */
/* be removed so the grandparent node must take over the pointer */
/* to the rest of the search tree */
if searchResults->prevDirection = DIRECTION_LEFT then
    modifiedGrandParentNode.leftBranch = removedParentNode.leftBranch
    modifiedGrandParentNode.leftLeaf = removedParentNode.leftLeaf
else
    modifiedGrandParentNode.rightBranch = removedParentNode.rightBranch
    modifiedGrandParentNode.rightLeaf = removedParentNode.rightLeaf
end if
```

```
/* overwrite grandparent node in the shadow copy and external SRAM  */
/* This write will update the search tree to remove the old parent  */
/* node and the old leaf in a single atomic operation               */
searchWriteSecondaryEntry(modifiedGrandParentNode,
                          modifiedGrandParentNodeAddr)

/* mark the secondary search address of the old parent node as  */
/* available for use */
releaseSecondaryAddress(removedParentNodeAddr)

return
}
```

6.8 Microprocessor Cell Interface Communicating

This section is a guide to programming the routines to communicate with the Input Microprocessor Cell Interface (IMCIF) and the Output Microprocessor Cell Interface (OMCIF). The application program will run these routines intermittently to insert cells into the cell stream and to respond to cell arrivals at the OMCIF.

The ATLAS-3200 has a one cell buffer on the IMCIF and a sixteen cell buffer on the OMCIF (see Figure 4) that are accessed through the indirect registers listed in Table 30. The reading and writing algorithms using these registers are discussed in section 7.10 MCIF Interfacing.

Table 30 - Registers for MCIF Indirect Accessing

Register	Description
Microprocessor Cell Interface Control and Status Register (0x020)	Indirect data read/write control bits.
Microprocessor Cell Data Register (0x021)	Indirect data register.

6.8.1 Reading Cells

Cells will be routed to the OMCIF depending on the type of cell and routing configuration bits. See Section 6.11 OAM Cell Processing for detailed information on how the cell routing configuration bits affect routing to the OMCIF. The reason that a cell was routed to the OMCIF can be determined by reading the causation word that is appended to cells if the Cell_Info_to_UP bit in the Cell Processor Configuration Register (0x100) is set.

The OMCIF has a 16 cell FIFO to buffer received cells. There are three ways to determine when there are cells available in the OMCIF:

- Poll the EXTCA bit in the Microprocessor Cell Interface Control and Status Register (0x020). When EXTCA is logic 1 there is at least one full cell in the FIFO.

- Use the UP_DMAREQ signal. This signal is asserted whenever EXTCA is asserted. The microprocessor can poll it or generate interrupts from it. The polarity of this signal is set by the DMAREQINV signal in the Microprocessor Cell Interface Control and Status Register (0x020).

- Use the INTB signal. Enable this interrupt signal for the OMCIF by setting the UPCAE bit in the Master Interrupt Enable #1 Register (0x004). The UPCAI bit

in the Master Interrupt Status #1 Register (0x002) is set whenever EXTCA goes high. See Section 6.10 Interrupt Handling for interrupt details.

Use the method that is most suitable for the application.

If a cell arrives at the OMCIF when the FIFO is full, the newly arrived cell will be discarded. This event triggers the UPOVRI bit in the Master Interrupt Status #1 Register (0x002) to be set. If this interrupt is enabled with the UPOVRE bit in the Master Interrupt Enable #1 Register (0x004) then the INTB signal will be asserted.

Note: *since there is no signal that indicates an almost full FIFO, the software should attempt to maintain an empty FIFO to prevent cell loss.*

See section 7.10.1 for the algorithm to read cells from the OMCIF using the indirect registers.

6.8.2 Writing Cells

The IMCIF allows the microprocessor to insert cells into a cell stream. The cells can be inserted either before or after the Cell Processor depending on the value of the PROC_CELL bit in the Microprocessor Cell Interface Control and Status Register (0x020). If they are inserted before the Cell Processor then they are treated as if they were coming from the Input Cell Interface (ICIF). If they are inserted in the cell stream after the Cell Processor they will not be searched, processed, or counted and will be output directly to the specified PHY.

Inserting cells into a cell stream is application specific so can be done as required. There are two major considerations that should be noted however:

- Cells should not be inserted too frequently to the IMCIF. The Cell Processor gives an equal priority to cells received from the ICIF and cells received from the IMCIF therefor it is the responsibility of the software to pace cell insertion on the IMCIF.

- Cells inserted with PROC_CELL = 1 it will be routed as if they came from the ICIF and may be routed to the OMCIF. For example if a cell is inserted to a VC that has the VC_to_UP set in its VC Table Record Configuration field the cell will arrive back at the OMCIF. The OMCIF has a 16 cell FIFO that should not be allowed to overflow to prevent cell loss. Cells that are inserted with PROC_CELL = 0 will never be routed to the OMCIF.

See section 7.10.2 for the algorithm to write cells to the IMCIF using the indirect registers.

6.9 FIFO Managing

The ATLAS-3200 provides two FIFOs that the microprocessor can optionally use to efficiently monitor the ATLAS-3200's operating status. The Change of State (COS) FIFO monitors all active VC's and records any changes in their states due to alarm conditions. The Count Rollover (CRO) FIFO monitors the saturating counters in the Performance Management and Cell Counting fields and records changes in the most significant bits. These two FIFO's each have a background process that continually scans the ATLAS-3200 for conditions that will trigger an entry. The use of these FIFOs relieves the microprocessor from having to poll a large number of fields in the data structures. If the microprocessor opts not to use these FIFOs then it is responsible for periodically reading each field that need to be monitored. This method is only feasible if there are a small number of connections in use.

The following sections discuss the operation and configuration of the COS FIFO and the CRO FIFO.

6.9.1 Change of State FIFO

The Change of State (COS) FIFO allows efficient monitoring of the connection states of all the active VCs. The state of a VC is stored in the Status field of its VC Table Record. The COS FIFO records changes that occur to the bits in the Status field. The status bits are divided into three types (DRAM CRC error, OAM Failure, and OAM Alarms) that can be selectively enabled for generating COS entries. The configuration bits described in Table 32 control which type of status bits cause entries to the COS FIFO.

The COS FIFO is 256 entries deep and maskable interrupts are provided to indicate when the FIFO is not empty, half full and full. The COS interrupt configuration bits are described in Table 31, and Section 6.10 Interrupt Handling contains further interrupt information.

Access to the FIFO entries are provided through the indirect access register listed in Table 33. See Section 7.11 for COS FIFO Interfacing algorithm.

To take advantage of this feature the microprocessor can maintain a shadow copy of each VC's status field in its own memory. The shadow copy can then be quickly read by all routines that need to reference a VC's status. When a status bit on the ATLAS-3200 changes, a entry will be sent to the COS FIFO and an interrupt will be caused if one of the interrupt conditions is met. The microprocessor can service the interrupts and read the COS FIFO entries then update its shadow copy. To ensure that the shadow copies are accurate the microprocessor should attempt to read the COS FIFO as frequently as possible,

and would ideally keep it empty. If the COS FIFO is not used then it is the responsibility of the software to poll each active VC Table Record to monitor the status bits.

Whether or not the COS FIFO is used, it is the responsibility of the management software to ensure that the status of each VC is read often enough so that changes in state monitoring remains compliant with the Bellcore and ITU standards.

Table 31 - Register Bits for COS FIFO Interrupts

Bit Name	Register	Description
COSI	Master Interrupt Status #1 (0x002)	Interrupt status flag indicating COS FIFO is not empty.
XCOSI	Master Interrupt Status #1 (0x002)	Interrupt status flag indicating COS FIFO is half full.
COSFULLI	Master Interrupt Status #1 (0x002)	Interrupt status flag indicating COS FIFO is full. The COS detection background process is halted.
COSE	Master Interrupt Enable #1 (0x004)	Enable COSI interrupt.
XCOSE	Master Interrupt Enable #1 (0x004)	Enable XFULLI interrupt.
COSFULLE	Master Interrupt Enable #1 (0x004)	Enable COSFULLI interrupt.

Table 32 - Register Bits for COS FIFO Configuration

Bit Name	Register or Field	Description
COS_EN	Cell Processor Configuration (0x100)	Enables the COS FIFO. If disabled no entries will be made to the COS FIFO. This bit supercedes all other COS bits.
COS_DRAM_ERR_EN	Cell Processor Configuration (0x100)	Controls the generation of COS FIFO entries caused by changes in the DRAM_CRC_Err bit in the VC Table Record, Status field. If set to logic 1 DRAM_CRC_Err changes will generate entries.

Bit Name	Register or Field	Description
[COS_Fail_En, COS_Fail_Only]	Cell Processor Configuration (0x100)	These control the generation of COS FIFO entries caused by changes in the OAM bits in the VC Table Record, Status field. Their actions are as follows: 00 : Only the changes to the OAM alarm bits (CC, AIS, and RDI alarms) are sent to the COS FIFO. Changes to OAM_Failure bit are not sent. 01 : No OAM change of states sent to the COS FIFO. 10 : Changes to all OAM bits sent to the COS FIFO. 11 : Only the changes to the OAM_Failure bit are sent to the COS FIFO. Changes to OAM alarm bits not sent. Use this if the software has no need to know about changes of connection state unless they rise to the level of a service failure (i.e. an OAM fault that persists for at least 4.5 +/- 0.5 seconds)

Bit Name	Register or Field	Description
COS_FIFO_enable	VC Table Record, Configuration Field	Enables COS entries on a per-VC basis. . 0 : Changes of bits in the Status field of this VC will not cause entries to the COS FIFO 1 : Changes of the Status field bits selected by [COS_DRAM_ERR_EN, COS_Fail_En, COS_Fail_Only], will cause entries to the COS FIFO for this VC.

Table 33 - Registers for COS FIFO Interfacing

Register	Address	Description
VC Table Change of Connection State FIFO Status	0x190	Status bits
VC Table Change of Connection State FIFO Data	0x191	Indirect data register

6.9.2 Count Rollover FIFO

The Count Rollover (CRO) FIFO allows the microprocessor to efficiently monitor the values of the internal counters and prevents information loss due to counter saturation. The CRO monitors selected count fields in VC Table Records, PM Table Records, Per-PHY Policing RAM Tables, and Per-PHY Counts. When the most significant bit (MSB) of a CRO enabled counter changes from 0 to 1, an entry will be entered into the CRO FIFO and the MSB will be reset to 0. The configuration bits described in Table 34 control which counters are enabled to cause entries to the CRO FIFO. Counts that are disabled from causing CRO FIFO entries will operate as normal saturating counters and it is the responsibility of the microprocessor to poll them frequently enough to prevent saturation. Counts that are designed to roll over in normal operation do not generate CRO FIFO entries.

The CRO FIFO is 256 entries deep and maskable interrupts are provided to indicate when the FIFO is not empty, half full and full. The CRO interrupt configuration bits are described in Table 35, and Section 6.10 Interrupt Handling contains further interrupt information.

Access to the FIFO entries is provided through the indirect access registers listed in Table 36. See Section 7.12 for the CRO FIFO Interfacing algorithm.

The intention is that the microprocessor uses these rollover entries to maintain the most-significant bits of the counters in its own memory. The least-significant bits can be accessed by the microprocessor through the indirect access registers whenever precise counts are needed. To ensure that accurate values are maintained and to prevent the FIFO from filling, the microprocessor should attempt to read the COS FIFO as frequently as possible, and should ideally keep it empty. If the Count Rollover FIFO is full, the MSB of the counts will remain set until such time as it can make an entry in the FIFO. The counts continue counting until they saturate. If the CRO FIFO is filling too quickly, the rate of CRO entries can be reduced by setting the Sat_Fast_PM_Counts bit in the Cell Processor Configuration Register (0x100). This prevents the counts in the Performance Management Table that can increment very quickly from generating Count Rollover FIFO entries, but will also likely result in these counts saturating and loss of information.

Table 34 - Register Bits for CRO FIFO Configuration

Bit Name	Register or Field	Description
CRO_FIFO_EN	Cell Processor Configuration (0x100)	Globally enables the CRO FIFO. If disabled no entries will be made to the CRO FIFO. All related count fields operate as normal saturating counters. This bit supercedes all other CRO bits.

Bit Name	Register or Field	Description
Sat_Fast_PM_Counts	Cell Processor Configuration (0x100)	Globally supercedes the PM Rollover_FIFO_EN bit for the fields that may saturate frequently. Disables, globally, the following fields in all PM Table Records from generating CRO FIFO entries: • Fwd Errors • Bwd Errors • Bwd Lost Fwd PM Cells • Bwd Lost Bwd PM Cells • Fwd Lost Fwd PM Cells. Set this if it is acceptable to let these fields saturate and it is desired to reduce the rate of entries into the CRO FIFO. If this is not set then the microprocessor must read the CRO FIFO frequently.
Rollover_FIFO_enable	VC Table Record, Configuration Field	Enables, per-VC, the following fields from the VC Table Record to generate CRO FIFO entries: • Count 1 • Count 2 • Alternate Count 1 • Alternate Count 2
Policing Rollover FIFO enable	VC Table Record, Policing Configuration Field	Enables, per-VC, the following fields from the VC Table Record to generate CRO FIFO entries: • Non-Compliant Count 1 • Non-Compliant Count 2 • Non-Compliant Count 3

Bit Name	Register or Field	Description
PHY Rollover_FIFO_EN	Per-PHY Counter Configuration (0x1A0)	Enables, globally, the following fields from the Per-PHY Counts to generate CRO FIFO entries: • Per-PHY CLP0 Cell Count • Per PHY CLP1 Cell Count • Per PHY Valid RM Cell Count • Per PHY Valid OAM Cell Count • Per PHY Errored OAM/RM Cell Count • Per PHY Invalid VPI/VCI/PTI Cell Count • Per-PHY EFCI/Non-Zero GFC Cell Count • Per-PHY Timed-Out Cell Count
PHY Policing Rollover FIFO EN	Per-PHY Policing RAM Table	Enables, per-PHY, the following fields from the Per-PHY Policing RAM to generate CRO FIFO entries: • Phy Non-Compliant1 • Phy Non-Compliant2 • Phy Non-Compliant3
PM Rollover_FIFO_EN	PM Table Record, Configuration Field	Enables, per PM Table Record, the following fields from the PM Table Record to generate CRO FIFO entries: • all fields in Rows 3,4,5,6, and 7, except FwdSECBC and Bwd SECBC Note that if the Sat_Fast_PM_Counts bit is set then it takes precedence over this bit and five of the fields will be disabled from generating entries.

Table 35 - Register Bits for CRO FIFO Interrupts

Bit Name	Register or Field	Description
CROI	Master Interrupt Status #1 (0x002)	Interrupt status flag indicating CRO FIFO is not empty.
XCROI	Master Interrupt Status #1 (0x002)	Interrupt status flag indicating CRO FIFO is half full.
CROFULLI	Master Interrupt Status #1 (0x002)	Interrupt status flag indicating CRO FIFO is full. The CRO detection background process is halted.
CROE	Master Interrupt Enable #1 (0x004)	Enable CROI interrupt.
XCROE	Master Interrupt Enable #1 (0x004)	Enable XFULLI interrupt.
CROFULLE	Master Interrupt Enable #1 (0x004)	Enable CROFULLI interrupt.

Table 36 - Registers for CRO FIFO Interfacing

Register	Address	Description
VC Table Count Rollover FIFO Status	0x198	Status bits
VC Table Count Rollover FIFO Data	0x199	Indirect data register

6.10 Interrupt Handling

This section contains a description of the interrupt hierarchy on the ATLAS-3200. This can be used as guide when programming the interrupt service routines.

The ATLAS-3200 contains two Master Interrupt Status registers and two corresponding Master Interrupt Enable registers. There are other interrupt related bits contained in various registers and data structures. The relationships between all the interrupt related bits are shown in Figure 26 and Figure 27.

Note that all the interrupts in the Master Interrupt Status #2 register will be disabled if the REG3E bit in the Master Interrupt Enable #2 register is set to logic 0.

Figure 26 - Interrupt Hierarchy, Interrupt Status #1 Reg.

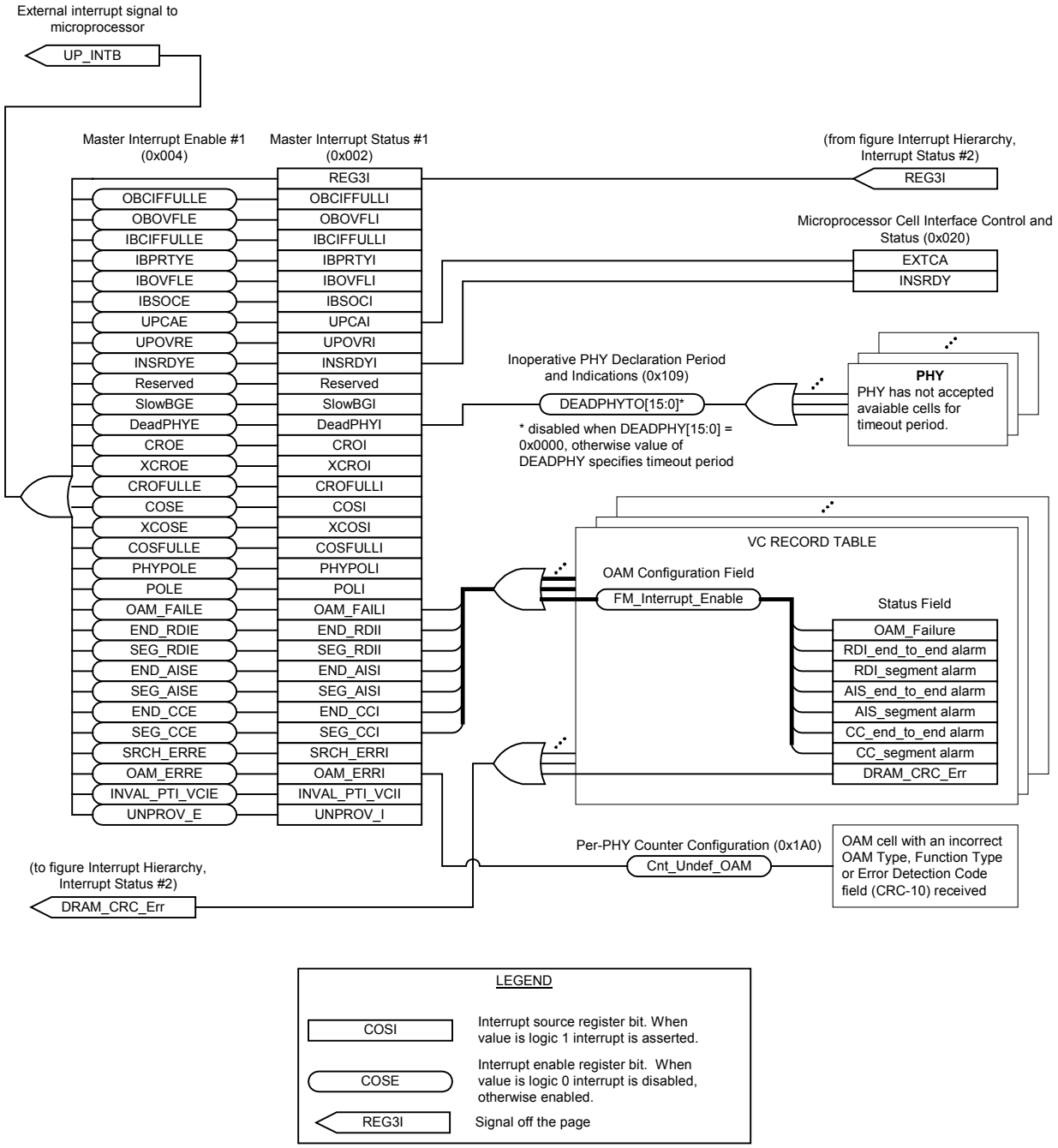
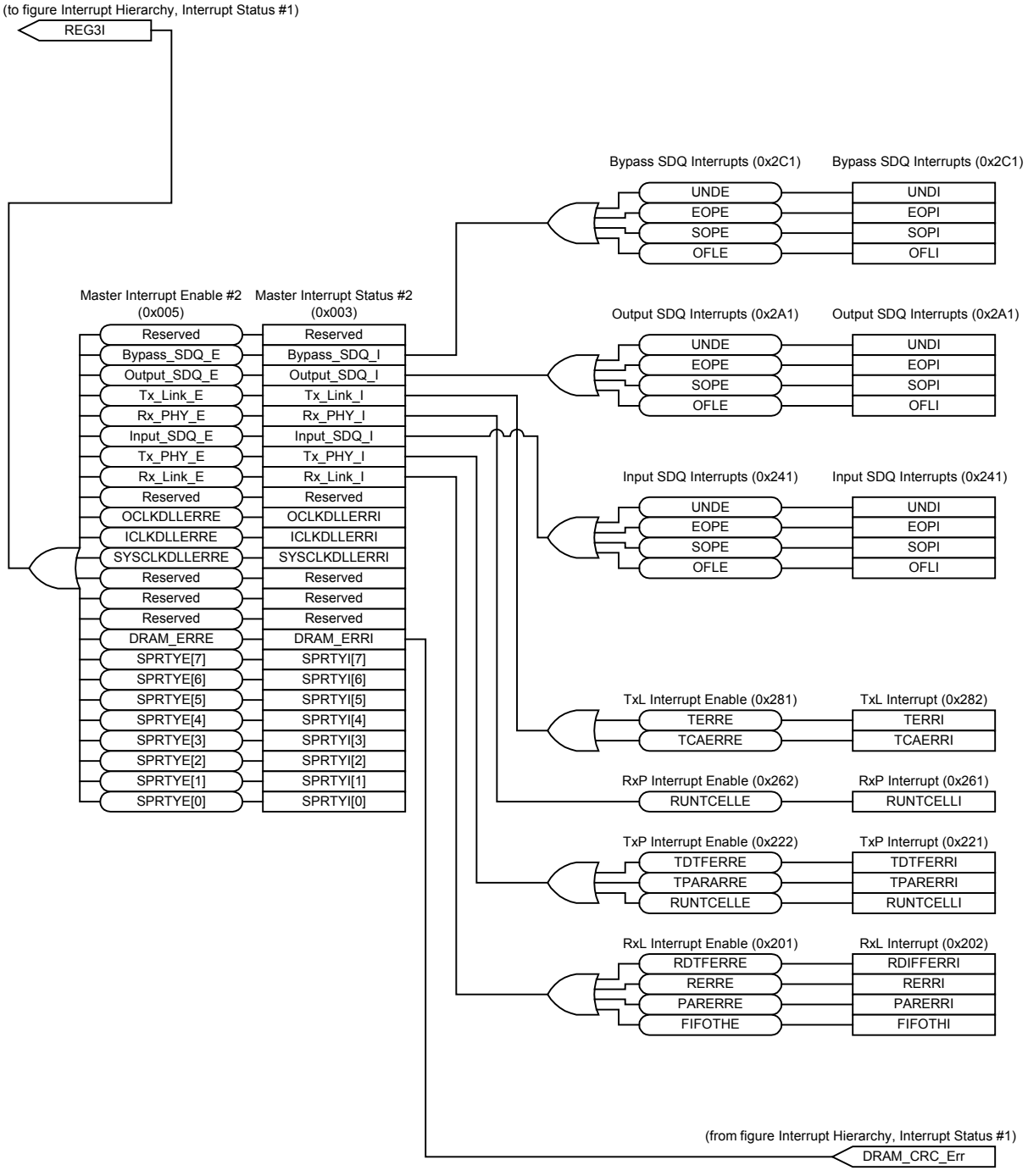


Figure 27 - Interrupt Hierarchy, Interrupt Status #2 Reg.



6.11 OAM Cell Processing

Operations and Maintenance (OAM) cell processing in the ATLAS-3200 is configured by global settings located in direct registers, by per-VC settings located in each VC Table Record, and by per-PM Session settings located in each PM Table Record. The global settings will typically be configured during ATLAS-3200 initialization, and the other settings will typically be configured as VCs or PM sessions are created.

The following sections describe the support provided by the ATLAS-3200 and the related configuration settings for each type of OAM cell.

For detailed information on OAM cell flow through the cell processor see Section 0, and for a general OAM reference see Appendix A: OAM Cell Descriptions.

6.11.1 General OAM Settings

The ATLAS-3200 contains some general OAM settings that are not specific to a certain OAM type. The settings listed in Table 37 configure general OAM operation for each VC. The settings listed in Table 38 configure general OAM operation for the all connections on the ATLAS-3200.

Table 37 - Per-VC General OAM Settings

Field in VC Table Record	Bit Name	Description
OAM Configuration	Generated OAM Defect Type [7:0]	Specifies the value of the 'Defect Type' field that is inserted into generated OAM cells. See Figure 49 for the location of the 'Defect Type' field within OAM cells.
OAM Configuration	F4toF5OAM	Controls whether or not an F5 (VCC) connection will send AIS or RDI cells due to AIS cells arriving on the associated F4 (VPC). This bit is only valid if this is a VCC. If this is a VPC this bit will have no effect.

Field in VC Table Record	Bit Name	Description
OAM Configuration	Segment Flow	Set this to 1 if the connection is part of a defined segment and is not a segment end-point.
OAM Configuration	Segment_End_Point	Set this to 1 if the connection is a segment termination point.
OAM Configuration	End_to_End_Point	Set this to 1 if the connection is an end-to-end termination point.
Segment Received Defect Type	[7:0]	When the ATLAS terminates an AIS cell, the defect location and type are extracted and placed in these fields. This information may be used to determine the location and nature of the fault. Additionally, the information is used for the defect location and type in RDI cells generated by the ATLAS using the AUTORDI function.
Segment Received Defect Location	[127:0]	
ETE Received Defect Type	[7:0]	
End-to-End Received Defect Location	[127:0]	

Table 38 - Global General OAM Settings

Register	Bit Name	Description
Cell Processor Configuration (0x100)	AUTO_AIS	Set this bit to enable automatic generation of AIS cells while in a Continuity alarm state. The AIS cells are inserted on a per-VC basis depending on each VC's CC_AIS_RDI bit.
Cell Processor Routing Configuration (0x101)	CRC10TOUP	Controls routing of OAM and RM cells that fail CRC check. See Section 6.12 Cell Routing for details.
Cell Processor Routing Configuration (0x101)	DROPCRCEOAM	
Cell Processor Routing Configuration (0x101)	DROPCRCERM	

Register	Bit Name	Description
Cell Processor Configuration (0x100)	GEN_HALFSECCLK	The 0.5 second clock is the clock that triggers background processing for some of the OAM functions, such as AIS, RDI and CC cell generation. This bit specifies whether the source is an internal or external cloack source. 0 : Background processing triggered on the rising edge of the external HALFSECCLK input 1 : Background processing triggered triggered from 0.5 second clock derived from SYSCLK, which is assumed to be 125 MHz

6.11.2 Fault Management Cell Processing

Fault Management OAM cells consist of Alarm Indication Signal (AIS), Remote Defect Indication (RDI), Continuity Check (CC), and Loopback (LB) cells.

The ATLAS-3200 supports cell generation, termination, and monitoring on a per-connection basis for segment and end-to-end F4 and F5 AIS, RDI and CC cells. For LB cells, support is provided for address identification, termination, and loopback on a per-connection basis.

The settings listed in Table 39 configure the per-VC Fault Management OAM operation. The settings listed in Table 40 configure the global Fault Mangement OAM operation that affects all connections on the ATLAS-3200.

Table 39 - Per-VC, Fault Management Settings

Field	Bit Name	Description
OAM Configuration	Send_AIS_segment	Set these bits to send segment or end-to-end AIS or RDI cells. The cells will be sent once per second until the appropriate bit is turned off. Send_RDI is not necessary if the AUTORDI function is enabled. Send_AIS would normally be used at non-end points to indicate failures downstream. The Generated_Defect_Type[7:0] field in the VC table is used for the defect type, and the Defect Location register fields are used for the defect location.
OAM Configuration	Send_AIS_end_to_end	
OAM Configuration	Send_RDI_segment	
OAM Configuration	Send_RDI_end_to_end	
OAM Configuration	CC_Activate_Segment	Enables Continuity Checking on segment and end to end flows. These bits enable the genereation of segement and user CC cells. The conditions for generating the cells depend on the setting of the bit ForceCC. If ForceCC is logic 0 then CC cells will be generated if no user cells arrive over a 1.0 second interval. If ForceCC is set to logic 1 then CC cells will be generated every 1.0 seconds regardless of user cell traffic.
OAM Configuration	CC_Activate_End_to_End	
OAM Configuration	FM_interrupt_enable	This bit enables the generation of segment and end-to-end AIS, RDI Continuity Check, and OAM Failure alarm interrupts. If this bit is logic 1, the ATLAS-3200 will assert the interrupts, as required, regardless of whether or not the ATLAS-3200 is a connection end-point (segment or end-to-end) for the connection. This bit would typically be programmed to logic 1

Field	Bit Name	Description
		at segment or end-to-end-points only. See Section 6.10 Interrupt Handling for further interrupt detail.
OAM Configuration	AUTO_RDI	Set this bit to automatically generate RDI cells while in AIS or CC Alarm.
OAM Configuration	CC_AIS_RDI	Enables AIS or RDI cells to be generated at one second intervals upon the declaration of a Continuity Alarm. The global configuration bit AUTO_AIS supercedes this bit, so AIS will not be generated in AUTO_AIS is logic 0.

Table 40 - Global Fault Management Settings

Register	Bit Name	Description
Cell Processor Configuration (0x100)	F4SAISF5EAIS	When this bit is logic 1, an end-to-end VC-AIS cell will be generated when a segment VPC-AIS cell is terminated at a VPC segment end-point.
Cell Processor Configuration (0x100)	F4SAISF5ERDI	When this bit is logic 1, an end-to-end VC-RDI cell will be generated when a segment VPC-AIS cell is terminated at a VPC segment end-point, and the VCC is also an end-to-end point.
Cell Processor Configuration (0x100)	F4EAISF5EAIS	When this bit is logic 1, an end-to-end VC-AIS cell will be generated when an end-to-end VPC-AIS cell is terminated at a VPC end-to-end point, and an associated VCC segment end-point is switched from that VPC.

Cell Processor Configuration (0x100)	F4EAISF5SRDI	When this bit is logic 1, a segment VC-RDI cell will be generated when an end-to-end VPC-AIS cell is terminated at a VPC end-to-end point and an associated VCC segment end-point is switched from that VPC.
Cell Processor Configuration (0x100)	ForceCC	This controls the mode of operation of continuity checking. 0 : CC cells are inserted only in periods of low user bandwidth. 1 : CC cells are inserted regardless of user bandwidth at a rate of once per second. The CC cells are inserted on a per-VC basis depending on each VC's CC_Activate_Segment or CC_Activate_End_to_End settings.
Cell Processor Routing Configuration (0x101)	LBtoOCIF	Control LB cell routing. See Section 6.12 Cell Routing for details.
Cell Processor Routing Configuration (0x101)	RTD_LB_TO_UP_AT_END	
OAM Defect Location (0x151, 0x152, 0x153, 0x154)	DL[128:0]	The defect location is inserted into non-automatic (ie, not as a result of AUTORDI) cell generation. The setting of these registers is system specific. It is expected that this would be set to some unique value within the network.
Per-PHY AIS Cell Generation Control (0x155, 0x156)	AIS[47:0]	Enables the generation of AIS cells, once per second, by all VC's on a given PHY. This is enabled on a per-PHY basis and supercedes all other AIS configuration bits. PHY on which this is disabled will generate AIS cells as normal.

Per-PHY RDI Cell Generation Control (0x157, 0x158)	RDI[47:0]	Enables the generation of RDI cells, once per second, by all VC's on a given PHY. This is enabled on a per-PHY basis and supercedes all other RDI configuration bits. PHY on which this is disabled will generate RDI cells as normal.
OAM Loopback Location ID Octets (0x160, 0x161, 0x162, 0x163)	LLID[128:0]	The loopback location ID that identifies this device in a network, to determine if arriving loopback cells should be processsed here. If a loopback cell carries a loopback Id that matches this one it may be looped back (Parent LB cell) or terminated (Return LB cell). See Section 6.12 Cell Routing for detailed LB cell flow information.

6.11.3 PM Cell Processing

The ATLAS-3200 supports highly configurable PM statistics maintenance. There are 512 individually configurable PM Table Records contained in two banks of internal PM RAM. The PM Table Records configure the PM operation and store performance information collected from received PM cells.

Each VC can link to two PM Table Records. Each VC Linkage Table Record contains two pointers that allow one PM Table Record from each bank to be addressed. The two PM Table Records can be used to perform simultaneous sinking and sourcing of a PM flow, simultaneous F4 and F5 PM flows, etc.. Multiple VC's can be linked to a single PM Table Record, and a VC can link or unlink from a PM Table Record without interrupting its operation.

The PM Table Records, however, do not track which VC's are linked to it, so the microprocessor should maintain its own data structure to track the PM Table Record usage. This data structure can be used to determine when a PM Table Record has no VC's linked to it and is thus available to be initialized for a new PM session.

The sections below discuss the configuration settings, and algorithms to initialize a PM Table Record, link a VC to a PM Table Record, and unlink a VC from a PM Table Record

6.11.3.1 Performance Management Configuration Settings

This is the only OAM type that is supported with additional data structures in memory. The PM Table Record data structures contain the configuration settings for each PM session. They are programmed indirectly through the indirect access registers listed in Table 41.

Table 41 - Registers for PM Table Indirect Accessing

Register	Address	Description
Performance Management RAM Record Address, Word Select and Access Control	0x170	Address field and control bits. See Section 7.3.3 for programming information.
Performance Management RAM Row 0 Word 0 (LSW)	0x171	Data registers corresponding to the rows of a PM Table Record. See Section 7.3.3 for programming information.
… (continues for Rows 1 to 6) …		
Performance Management RAM Row 7 Word 2 (MSW)	0x189	

The configuration settings of the PM Table Records are described in detail in Table 42. The global settings that configure the global operation for all PM sessions on the ATLAS-3200, are located in direct registers, and are listed in Table 43.

Table 42 - PM Table, Config. and Status Field

Bit Name	Description
Source_FwdPM	Bits 15:14 have the following mapping:
Generate_BwdPM	00 : Source neither forward or backward PM cells. 01 : Source backward PM cells upon receipt of forward PM cells. 10 : Source forward PM cells 11 : Reserved
F4_F5B	0 : This PM session is for a VCC (F5 OAM flow). 1 : This PM session is for a VPC (F4 OAM flow).
ETE_SegB	0 : This PM session is for a Segment flow. 1 : This PM session is for an End-to-End flow.

Bit Name	Description
Force_FwdPM	If Source_FwdPM = 1 then: 0 : Insertion of Fwd Monitoring cells governed by settings in Fwd PM Pacing and Head of Line Blocking Register (0x108). 1 : Forward PM pacing is ignored. Cell insertion forced when PM Table Record field “Current Count CLP0+1” = 1.5 * Blocksize
Threshold_Select[1:0]	Selects one of the global threshold configurations 00 : Performance Management Threshold A Register (0x189) 01 : Performance Management Threshold B Register (0x18A) 10 : Performance Management Threshold C Register (0x18B) 11 : Performance Management Threshold D Register (0x18C)
Blocksize[3:0]	Selects the Performance Management block size. After every block of user cells a PM cell will be inserted. Use the encoding given in the Data Sheet [1]. Choose this value based on the expected user cell data rate, the desired time interval between PM cell transmission and the acceptable portion of bandwidth that can be allocated for non-user cells. Small block sizes are not recommended for active connections as the PM cells could amount to a significant portion of the available user bandwidth.
CLP0_SECBs_Only	0 : Severely Errored Cell Blocks (SECBs) will be declared due to lost cells whose CLP = 1. 1 : Severely Errored Cell Blocks (SECBs) will not be declared due to lost cells whose CLP = 1. Use this for connections on which there is no service guaranteee for CLP = 1 cells.
PM Rollover FIFO Enable	Enables the fields in the PM Table Record to generate CRO FIFO entries. See Section 6.9.2 Count Rollover FIFO for detailed CRO FIFO information.

Bit Name	Description
	0 : Count fields will saturate at the maximum value. Use this setting if the Count Rollover FIFO is not being used. The Cell Count fields must be polled periodically. 1 : CRO enabled fields will generate an entry in the Count Rollover FIFO and rollover when their MSBs are set. Use this setting if the Count Rollover FIFO is being used.
Reserved	Always set to logic 0 when writing. Read value is undefined.
Fwd_PM0	If Source_FwdPM is set to logic 0 then the Fwd_PM0 bit must be set to logic 1 initialily. This bit is cleared upon receiving the first Forward Monitoring cell, along with the current cell count, BIP-16, and the entire contents of rows 3 and 4. The Fwd_PM0 bit is used to denote the arrival of the first Forward Monitoring cell. The Fwd_PM0 bit suppresses accumulation of the Forward error counts. If this bit is not set, error counts will be accumulated. If Source_FwdPM is a logic 1, then if this bit is set to a logic 1 initially, rows 1 and 7 will be cleared at the end of the first block of user cells. Initializing Row 0 is the responsibility of the management software during setup.
Bwd_PM0	This must be set to a logic 1 initially. This bit is cleared upon receiving the first Backward Reporting cell. At that time the contents of rows 5, 6, and 7 are cleared (except for the Bwd SECBC count which is copied from the Backward Reporting cell) and Row 2 is initialized with values copied from the Backward Reporting cell.

Table 43 - Global Performance Management Settings

Register	Bit Name	Description
Cell Processor Configuration (0x100)	Copy_FwPM_Timestamp	Controls timestamp setting in generated Bwd PM cells. When the Copy_FwPM_Timestamp bit is logic 1, then If, or if this bit is logic 0, then the timestamp of the Bwd PM cell is set to the default all-ones. 0 : Timestamp of Bwd PM cells are set to the default all-ones. 1 : When a Bwd PM cell is generated to the BCIF upon reception of a Fwd PM cell and the BCIF is not full then the timestamp of the generated Bwd PM cell is set equal to the timestamp of the Fwd PM cell. If the BCIF is full when the Fwd PM cell arrives timestamp of Bwd PM cells are set to the default all-ones since there may be a delay before the Bwd PM is generated.
Fwd PM Pacing and Head of Line Blocking (0x108)	FPMP[15:0]	Forward PM Pacing control. Specifies the number of cells between insertion of consecutive Fwd Monitoring PM cells for all PM sessions. Set this with consideration to the activity of the connection and effects of the PM cell load. This should be programmed during ATLAS-3200 Initialization. Note that the Fwd Pacing can be disabled for individual PM sessions with the Force_FwdPM bit in the PM Table Record.

Register	Bit Name	Description
	FPMTO[15:0]	Forward PM Time Out control. Specifies the number of cell periods that a Fwd PM cell can wait to be inserted before being discarded. This feature will prevent malfunctioning PHY's from blocking the PM flows on the other PHY's. This should be programmed during ATLAS-3200 Initialization. 0x0000 : Time Out is disabled and Fwd PM cells can wait indefinitely. This is the default setting.
Performance Management Threshold A, B, C, D (0x189, 0x18A, 0x18B, 0x18C)	MMISINS[11:0] MERROR[3:0] MLOST[3:0]	Thresholds of misinserted, errored, and lost cells per Performance Management block required to declare a Severely Errored Cell Block (SECB) for misinserted cells (SECB Misinserted) on a forward or backward PM flow. When a SECB is declared on a PM flow, then either the SECB Misinserted, SECB Errored, or SECB Lost fields in the PM Table Record will be incremented. For example, if MMISINS[11:0] = 0x10A then 266 misinserted cells would have to arrive in a single PM block to increment the Fwd or Bwd SECB Misinserted Count (depending on flow direction) field. There are four sets of these fields, in the A, B, C, and D registers. Each PM Table Record uses its Threshold_Select field to choose one set of these threshold configurations for its PM session. These threshold values will typically be programmed during ATLAS-3200 Initialization. If set to all zeroes then detection for that type of SECB is disabled and the corresponding field in the PM Table Record will not be incremented.

6.11.3.2 Algorithm for PM Table Record Initializing

To create a new Performance Management session a free PM Table Record needs to be initialized with the desired settings. The general algorithm is as follows:

1. Select either PM Bank 1 or PM Bank 2 as the location for the PM Table Record.

2. Locate an unused PM Table Record.

3. Write to Row 0 of the PM Table Record with the PM Configuration & Status Field set to the values described in Table 42 and the remaining fields set to all zeroes. See section 7.3.3 for the write procedure.

   Ensure that Fw_PM0 and Bw_PM0 in the PM Configuration & Status Field are set to 1 so that Rows 1 to 7 are initialized automatically. When the first block of user cells is received, Rows 1 and 7 are cleared and Fw_PM0 is set to 0. When the first Backward Reporting cell is received, Rows 5, 6, and 7 are cleared (except for the Bwd SECBC count which is copied from the Backward Reporting cell), Row 2 is initialized with values copied from the Backward Reporting cell, and Bw_PM0 is set to 0.

6.11.3.3 Algorithm for PM Table Record Linking

A Virtual Connection can link to an initialized PM Table Record using the following procedure:

1. Select an initialized PM Table Record from either Bank 1 or Bank 2. The PM Table Record may already be in use by another VC. Note that each VC can link to one PM Table Record in Bank 1 and one in Bank 2.

2. Write to the VC's VC Linkage Table Record with the following settings, choosing either bank 1 or 2 fields depending on which bank the PM Table Record is in:

   - PM 1 Address/PM 2 Address set to the PM Table Record's address
   - PM 1 Active/PM 2 Active set to logic 1

3. Update the microprocessor data structure that tracks the VC's linked to each PM Table Record.

6.11.3.4 Algorithm for PM Table Record Unlinking

A Virtual Connection can unlink from a PM Table Record using the following procedure:

1. Write to the VC Linkage Table Record and set the PM 1 Active/PM 2 Active bit to 0, choosing either bank 1 or 2 field depending on which bank the PM Table Record is in.

2. Update the microprocessor data structure that tracks the VC's linked to each PM Table Record. If there are no more VC's linked to this PM Table Record then mark it as an available table.

6.11.4 APS Cell Processing

The contents of Automated Protection Switching (APS) cells are not processed by the ATLAS-3200 but they have configurable routing through the cell processor. The settings listed in Table 44 configure Automated Protection Switching OAM operation for the all connections on the ATLAS-3200.

Table 44 - Global APS Settings

Register	Bit Name	Description
Cell Processor Routing Configuration (0x101)	APSTOBCIF	Control APS cell routing. See Section 6.12 Cell Routing for details.
Cell Processor Routing Configuration (0x101)	APSTOOCIF	
Per-PHY APS Indication 1 (0x159)	APS[47:0]	Enables APS switching on a per-PHY basis. If APS switching is enabled for a PHY, then when a segment VP-AIS cell is terminated on it, an end-to-end VP-AIS cell will not be generated. If APS switching is disabled for a PHY, then when a segment VP-AIS cell is terminated, an end-to-end VP-AIS cell may be generated.
Per-PHY APS Indication 2 (0x15A)		

6.11.5 Activation/Deactivation Cell Processing

The contents of Activation/Deactivation (A/D) cells are not processed by the ATLAS-3200 but they have configurable routing through the cell processor. The settings listed in Table 45 configure Activation/Deactivation OAM operation for the all connections on the ATLAS-3200.

Table 45 - Global Activation/Deactivation Settings

Register	Bit Name	Description
Cell Processor Routing Configuration (0x101)	ACTDETOBCIF	Control A/D cell routing. See Section 6.12 Cell Routing for details.
Cell Processor Routing Configuration (0x101)	ACTDETOUP	
Cell Processor Routing Configuration (0x101)	ACTDETOOCIF	

6.11.6 System Management Cell Processing

The contents of System Management (SM) cells are not processed by the ATLAS-3200 but they have configurable routing through the cell processor. The settings listed in Table 46 configure System Management OAM operation for the all connections on the ATLAS-3200.

Table 46 - Global System Management Settings

Register	Bit Name	Description
Cell Processor Routing Configuration (0x101)	SYSMANTOBCIF	Control SM cell routing. See Section 6.12 Cell Routing for details.
Cell Processor Routing Configuration (0x101)	SYSMANTOUP	
Cell Processor Routing Configuration (0x101)	SYSMANTOOCIF	

6.11.7 Resource Management Cell Processing

The contents of Resource Management (RM) cells are not processed by the ATLAS-3200 but they have configurable routing through the cell processor. The

settings listed in Table 47 configure System Management OAM operation for the all connections on the ATLAS-3200.

Table 47 - Global Resource Management Settings

Register	Bit Name	Description
Cell Processor Routing Configuration (0x101)	DropRM	Control RM cell routing. See Section 6.12 Cell Routing for details.
Cell Processor Routing Configuration (0x101)	RMtoBCIF	
Cell Processor Routing Configuration (0x101)	RMtoUP	

6.12 Cell Routing

ATM cells are routed through the ATLAS-3200's cell processor based on properties of the cells and on the setting of the various cell routing configuration bits. This section provides a reference to the cell routing with flow charts illustrating how each type of ATM cell is affected by the cell routing configuration bits. Note that the flow charts are only intended to explain the cell routing configuration bits and do not show cell termination or cell generation.

There are global cell routing configuration bits that affect cells on all connections, and there are per-VC routing configuration bits that affect cells only for a specific VC. The global configuration bits will typically be set during ATLAS-3200 initialization (see Section 6.2 ATLAS-3200 Initializing), and the per-VC configuration bits will typically be set each time a VC is added (see Section 6.5 Virtual Connection Adding).

There are three sources for cells in the ATLAS-3200:

- Input Cell Interface (ICIF)
- Input Microprocessor Cell Interface (IMCIF)
- Input Backwards Cell Interface (IBCIF)

The incoming cells are passed through the Cell Processor for processing and are then directed them to any of the three cell outputs:

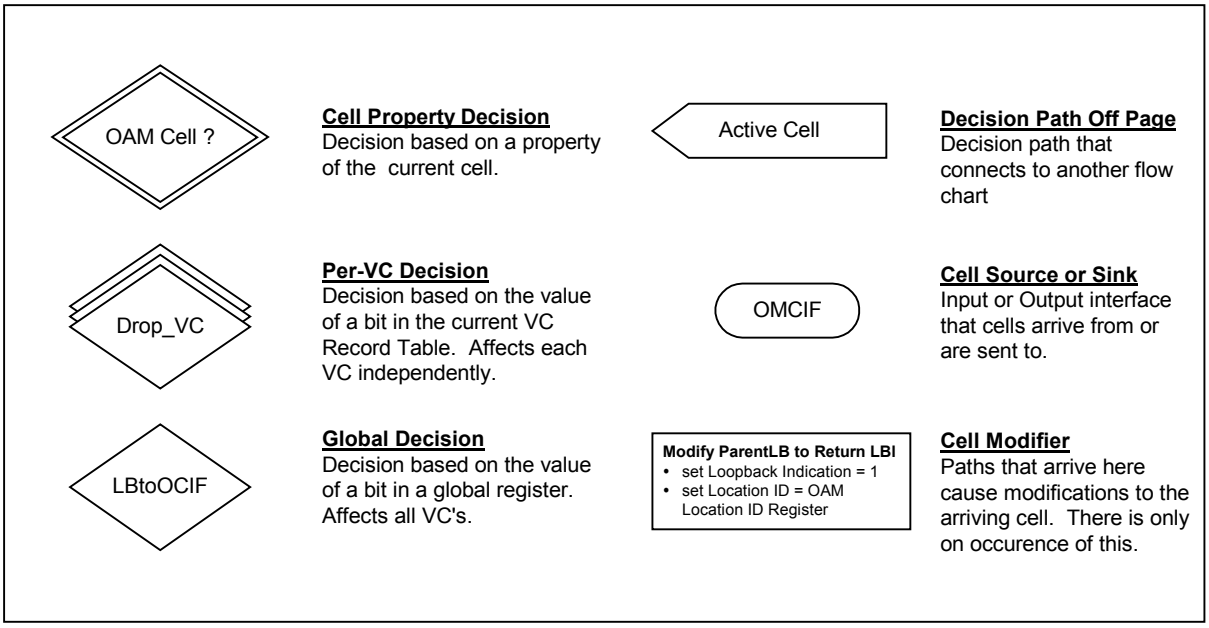
- Output Cell Interface (OCIF)
- Output Microprocessor Cell Interface (OMCIF)
- Output Backwards Cell Interface (OBCIF)

Although OAM cells are typically only a small portion of the total cell flow, the majority of the cell routing configuration is devoted to handling OAM flows. For background OAM information, see Appendix A: OAM Cell Descriptions for a reference to the OAM cell structures and their intended uses.

6.12.1 Flow Chart Guide

Flow charts are used to illustrate how the cell processor routes cells based on cell properties, per-VC configuration bits, and global configuration bits. The symbols shown in Figure 28 are used in the flow charts.

Figure 28 - Cell Flow, Legend



On the flow charts, some decision boxes do not have both possible decision paths shown. If a decision path is not shown it means that decision stream that reached that box is terminated and no further paths will be taken from there. However, any other streams on the flow diagram will still continue.

Converging paths represent an ‘OR’ connection. If one or more of the converging paths is followed then the decision stream will continue on the subsequent path. Note that if more than one converging paths are followed there is still only one copy of the cell.

As an example, in Figure 39 starting at the ‘YES’ path from the ‘A/D cell?’ decision, the paths to the OCIF, OMCIF, and OBCIF will be determined by the subsequent decision boxes as detailed in Table 48, Table 49, and Table 50.

Table 48 - Example Cell Flow to OCIF Logic Chart

ACTDEtoOCIF	Cell at flow end point?	Drop_VC	Result
0	YES	X	Cell will NOT go to the OCIF Neither of the converging paths to the Drop_VC decision box is followed so the cell can't reach the OCIF.
1	X	0	Cell will be copied to the OCIF. A path to the Drop_VC will always be reached regardless of the value of 'Cell at flow end point?'
X	NO	0	Cell will be copied to the OCIF. A path to the Drop_VC will always be reached regardless of the value of ACTDEtoOCIF.
X	X	1	Cell will NOT go to the OCIF. At the Drop_VC decision box, a value of 1 indicates that no further paths will be taken from here. Cells are not prevented from reaching the OMCIF or OBCIF however.

Table 49 - Example Cell Flow to OMCIF Logic Chart

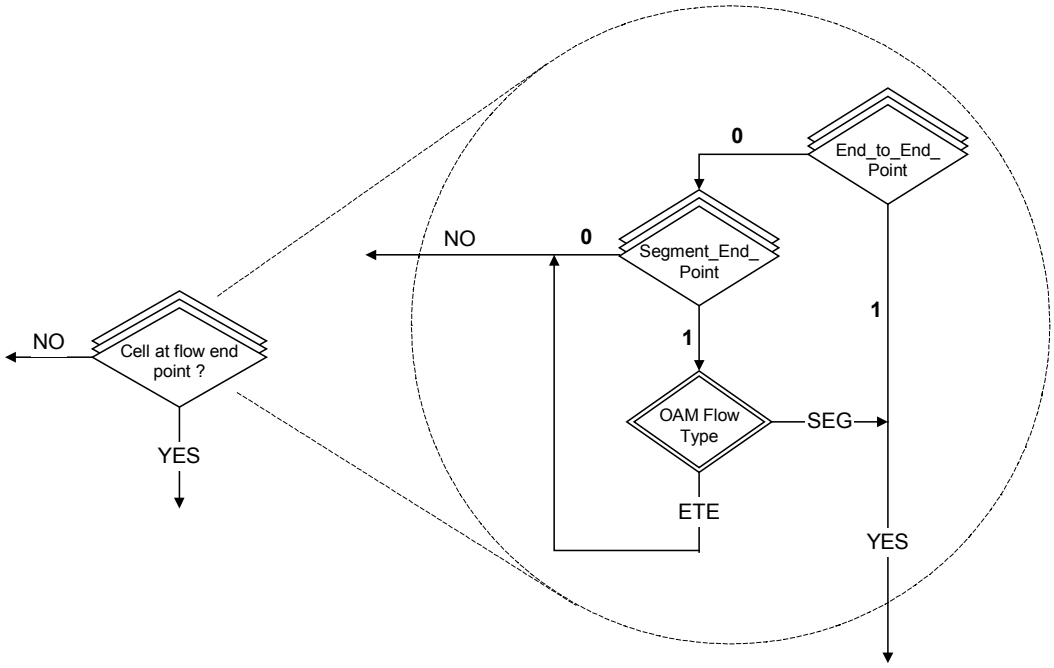
Cell at flow end point?	ACTDEtoUP	Result
YES	1	Cell will be copied to the OMCIF
X	0	Cell will NOT go to the OMCIF.
NO	X	Cell will NOT go to the OMCIF.

Table 50 - Example Cell Flow to OBCIF Logic Chart

Cell at flow end point?	ACTDEtoBCIF	Result
YES	1	Cell will be copied to the OBCIF
X	0	Cell will NOT go to the OBCIF.
NO	X	Cell will NOT go to the OBCIF.

One decision that frequently determines the routing of an OAM cell is whether or not the cell is at a flow end point. A cell is at a flow end point either when it is a Segment cell and it reaches a Segment End Point, or when it reaches an End-To-End Point regardless of whether it is a Segment or End-to-End cell. The symbol shown in Figure 29 is used to represent this decision in other flow charts. Note that it is a decision based on two per-VC configuration bits, and one cell property.

Figure 29 - Cell at Flow End Point Symbol

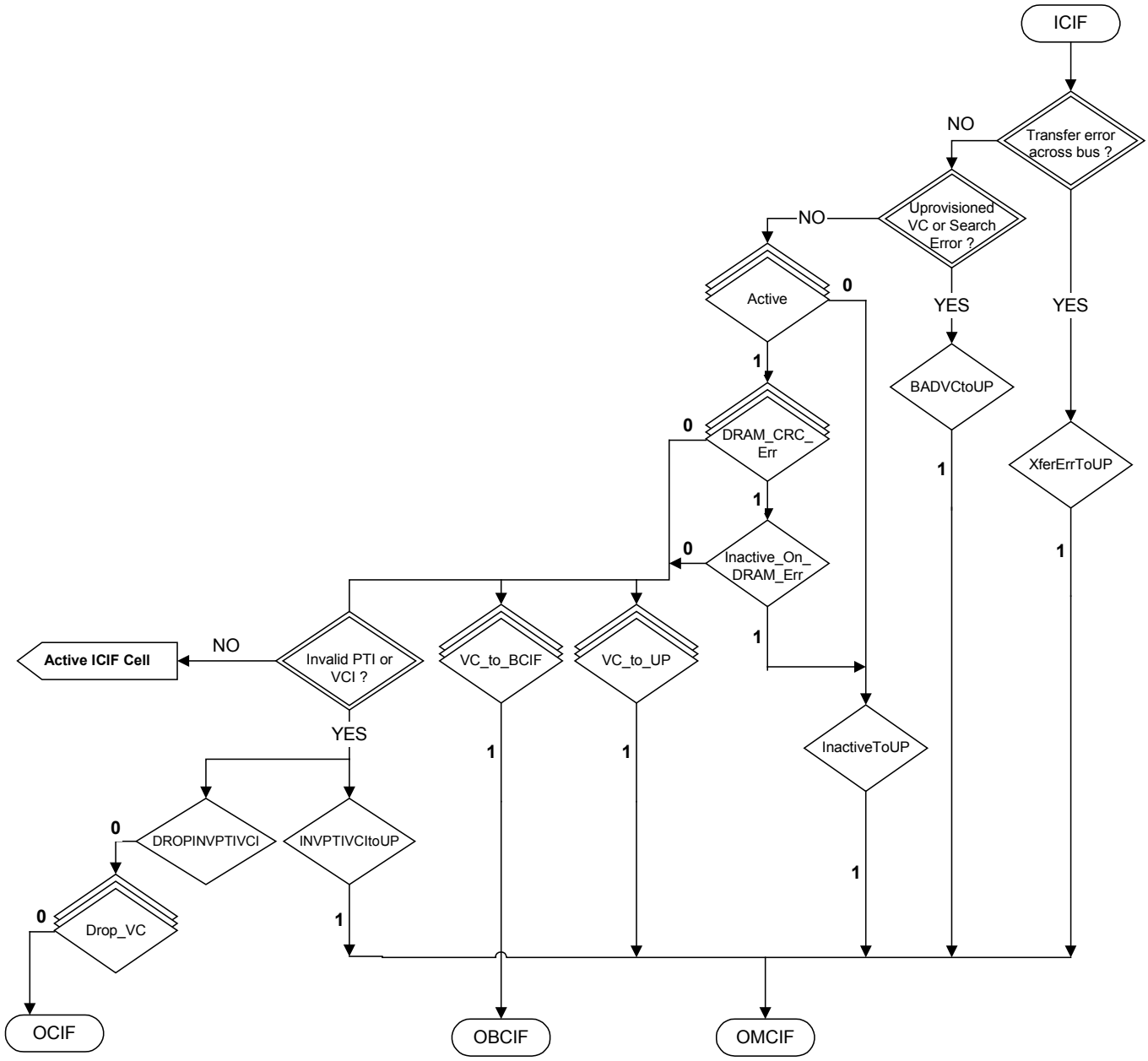


6.12.2 IMCIF Cell Routing

When a cell is inserted by the microprocesor to the Input Microprocessor Cell Interface (IMCIF), the routing of the cell depends whether the PROC_CELL bit was set for that cell. If PROC_CELL is set to logic 1, then the cell will be processed as if it originated from the Input Cell Interface (ICIF) and will follow the Cell Routing flow charts for the ICIF in Section 6.12.3. If PROC_CELL is set to logic 0 then the cell will simply proceed directly to the OCIF without being routed through the cell processor.

6.12.3 ICIF Cell Routing

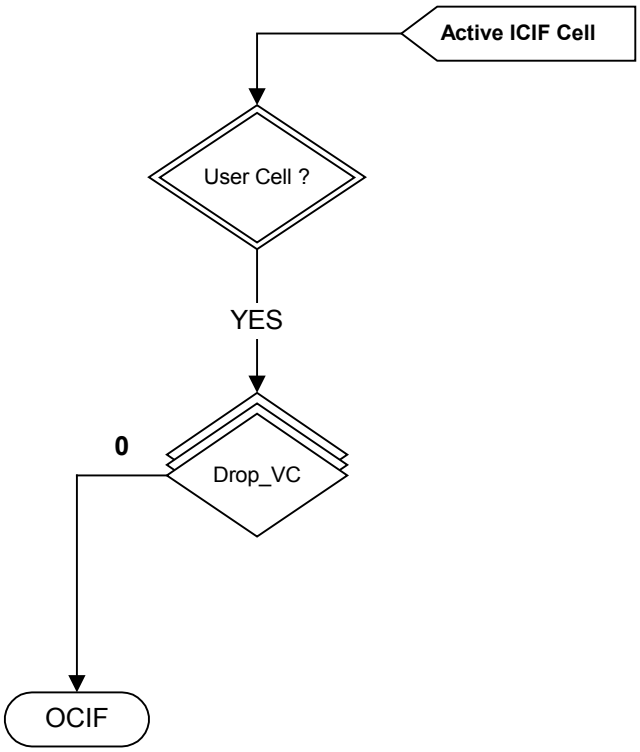
Figure 30 - Preliminary Cell Flow from ICIF



NOTE: *Cells may be routed to the OBCIF or OMCIF (by VC_to_BCIF and VC_to_UP) here, regardless of any future decisions on 'Active ICIF Cell'.*

6.12.3.1 User Cell Flow from ICIF

Figure 31 - User Cell Flow



6.12.3.2 OAM Cell Flow from ICIF

Figure 32 - Preliminary OAM Cell Flow

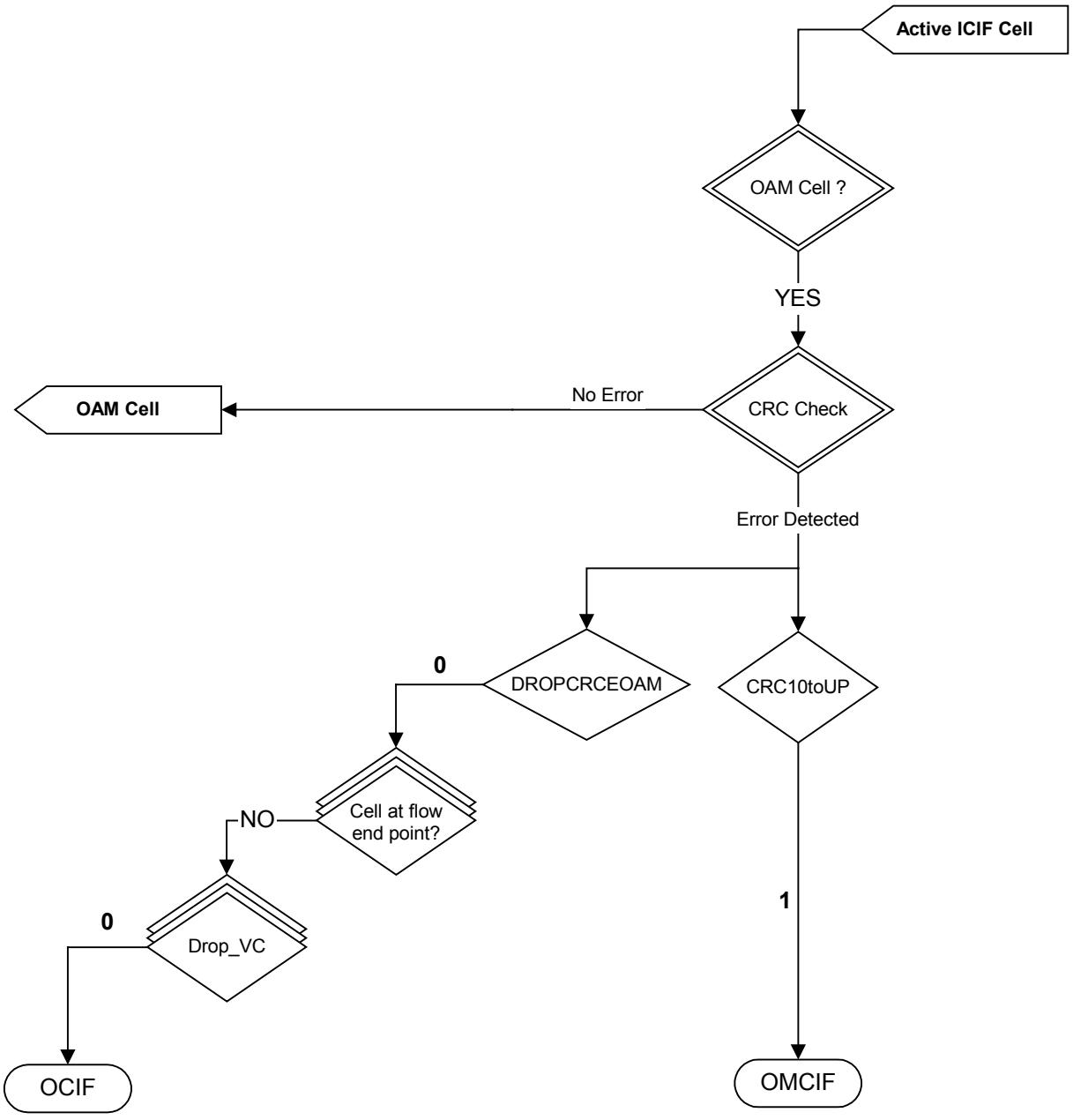


Figure 33 - AIS/RDI/CC Cell Flow

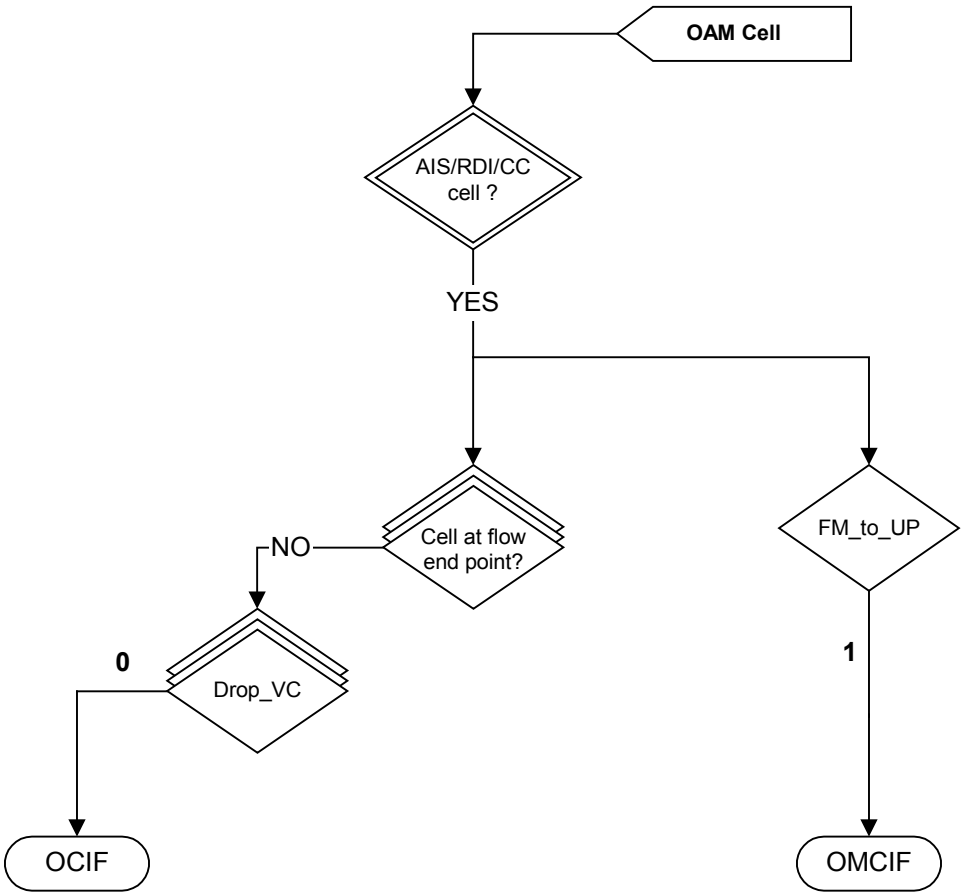


Figure 34 - Loopback Cell Flow, (LB_Route = '00')

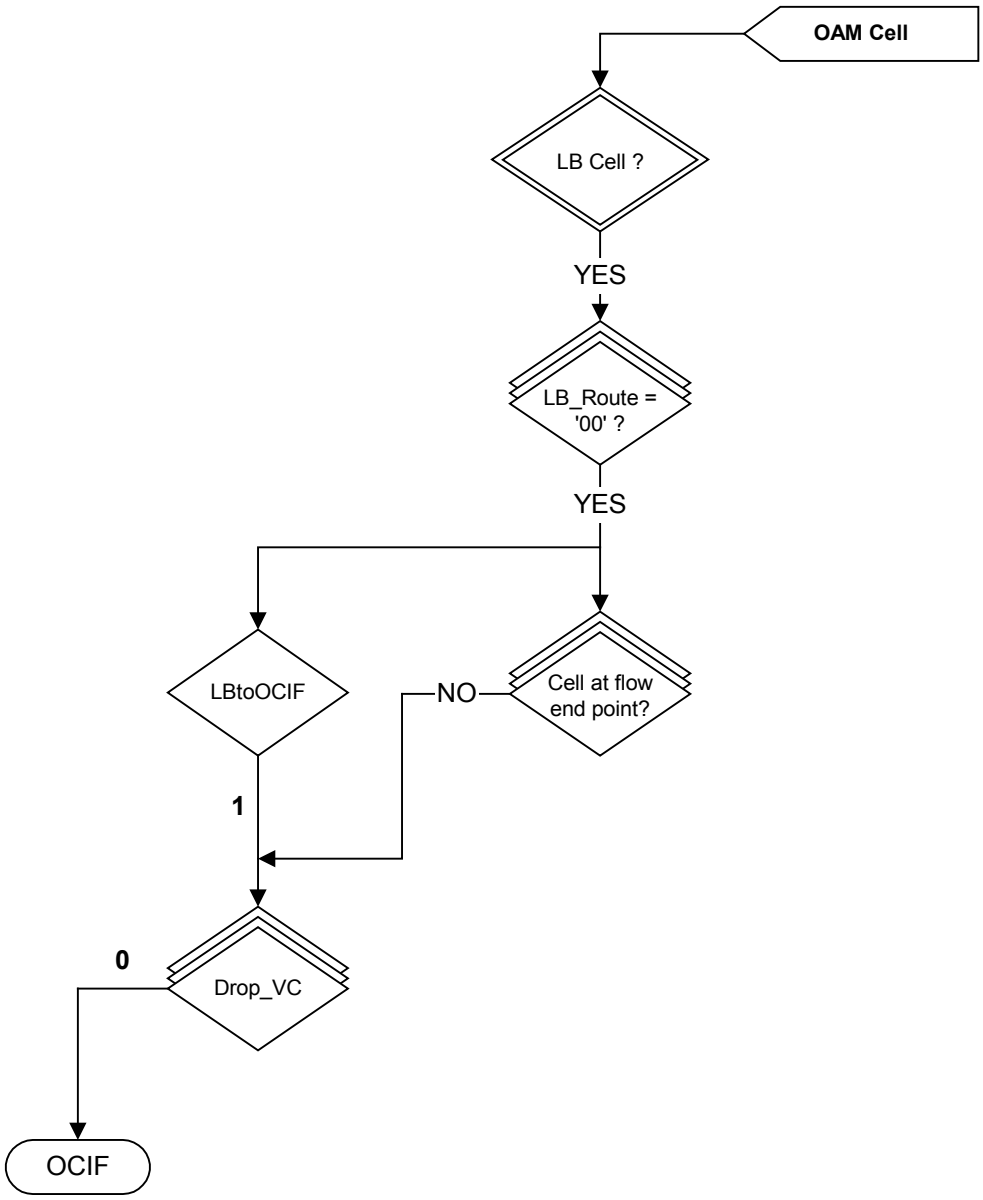


Figure 35 - Loopback Cell Flow (LB_Route='01' or '10',Parent)

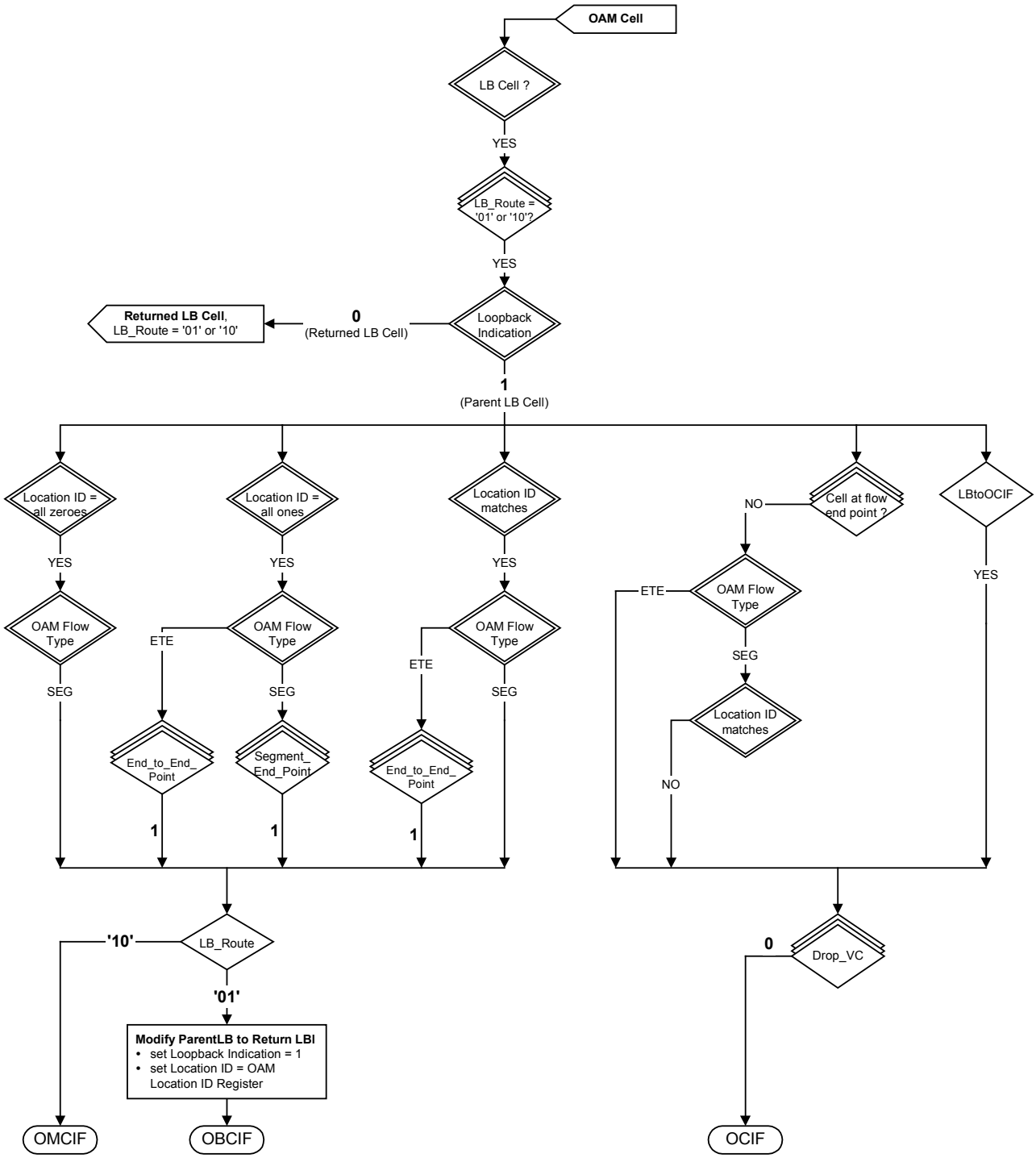


Figure 36 - Loopback Cell Flow (LB_Route='01' or '10', Return)

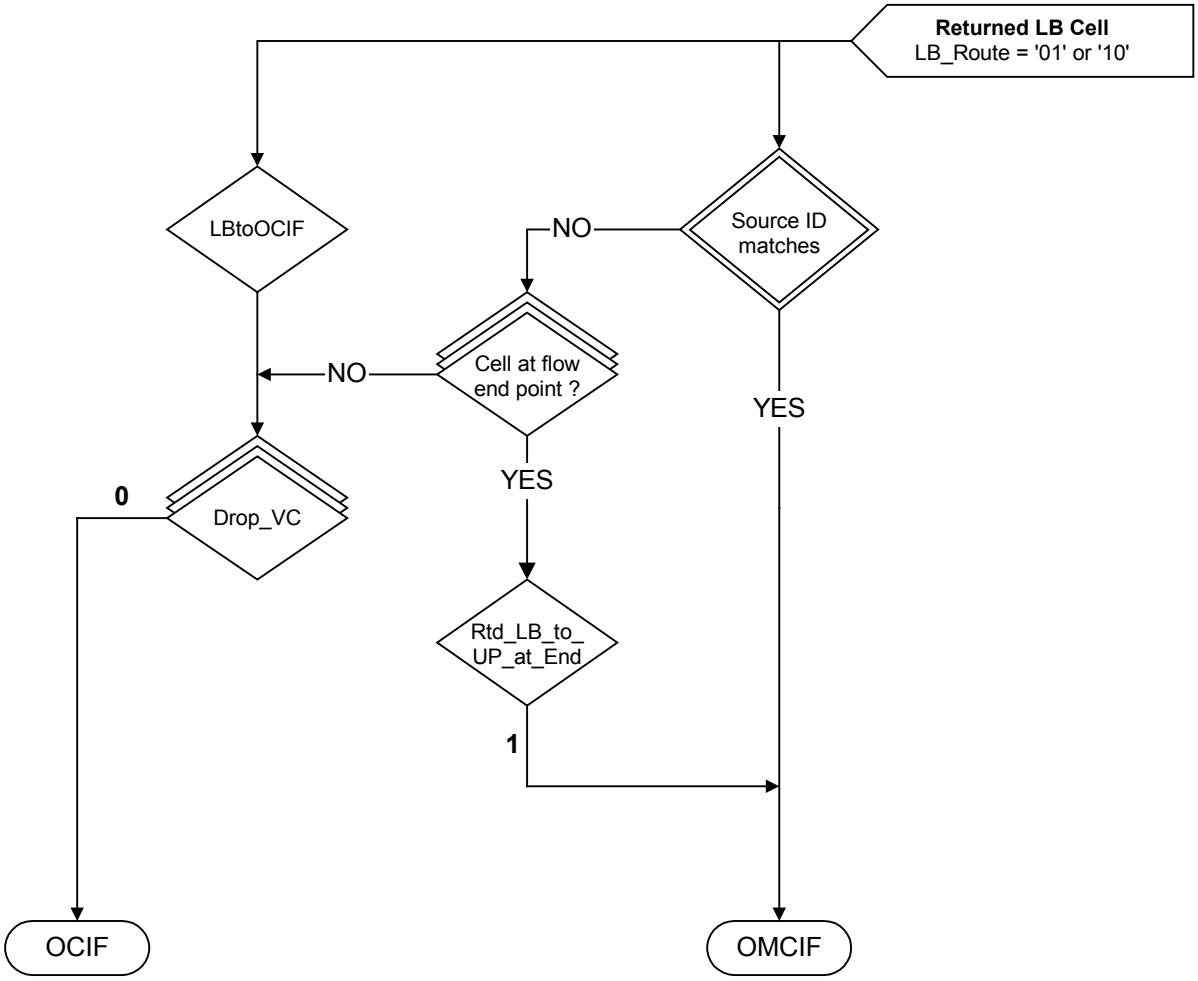


Figure 37 - Loopback Cell Flow, (LB_Route = ‘11’)

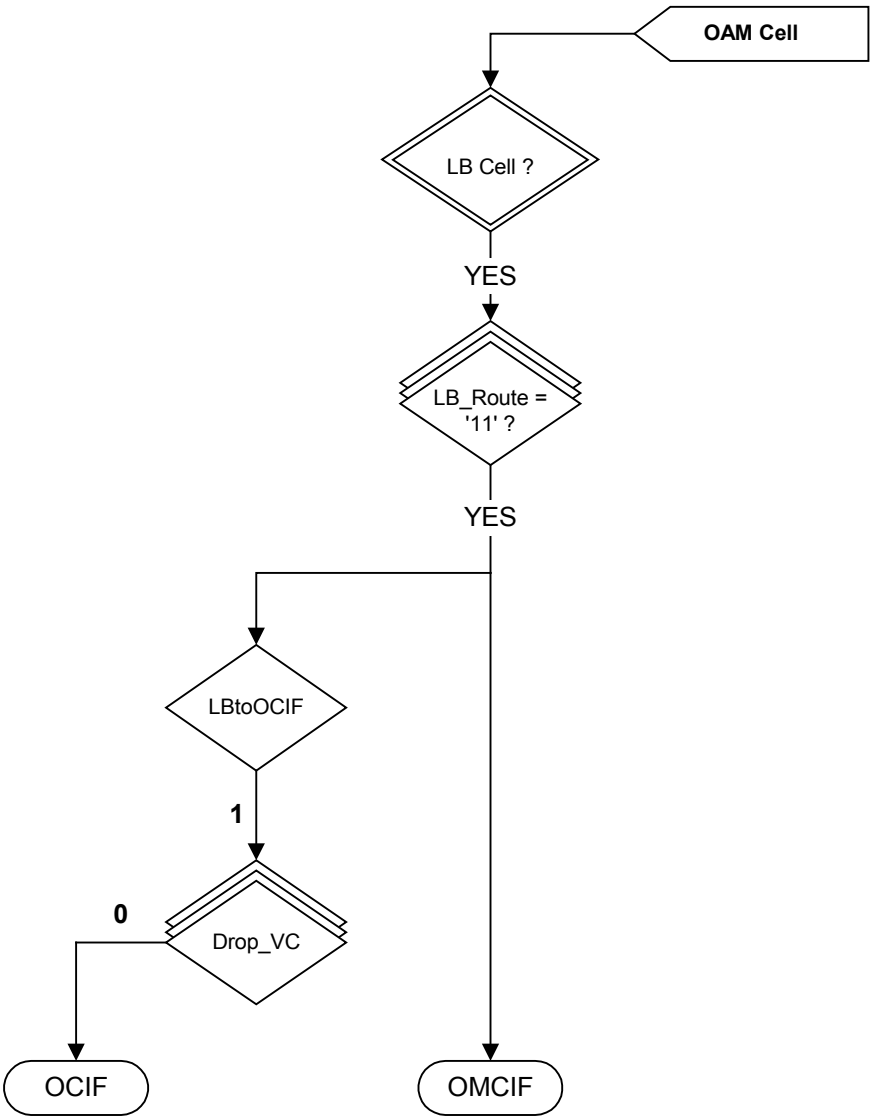


Figure 38 - PM Cell Flow

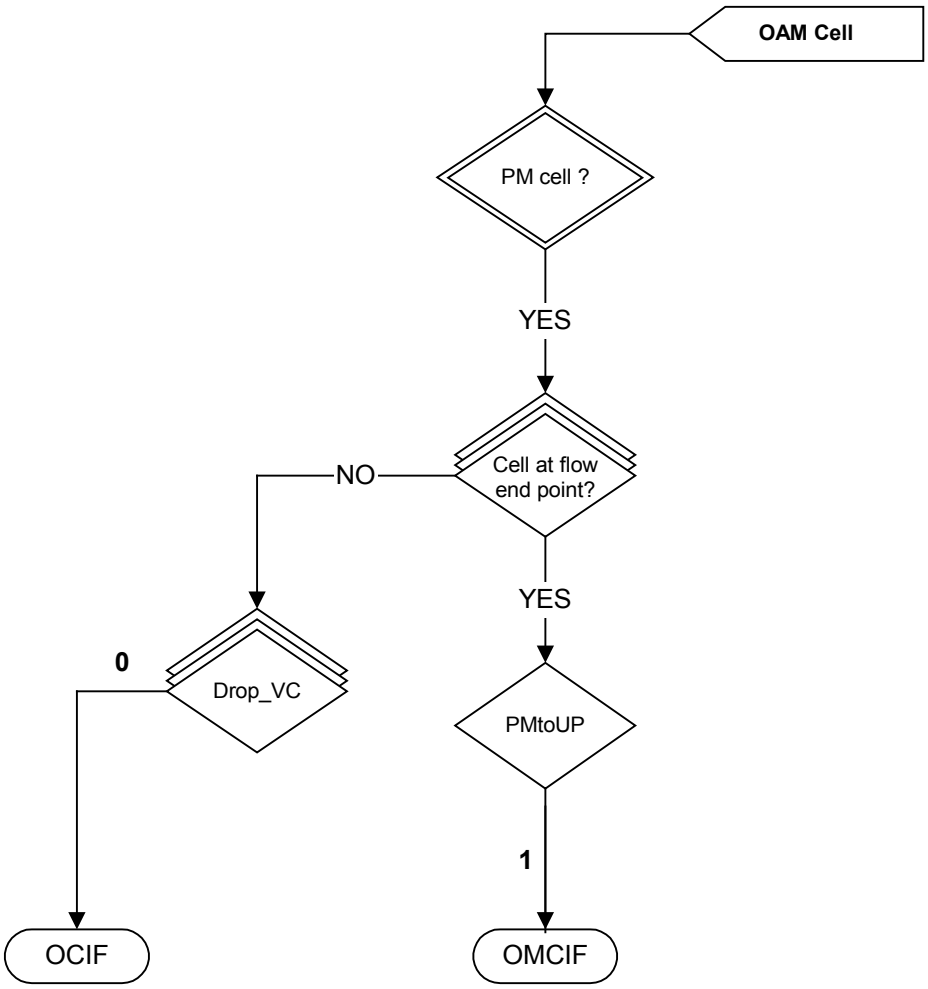


Figure 39 - Activate/Deactivate Cell Flow

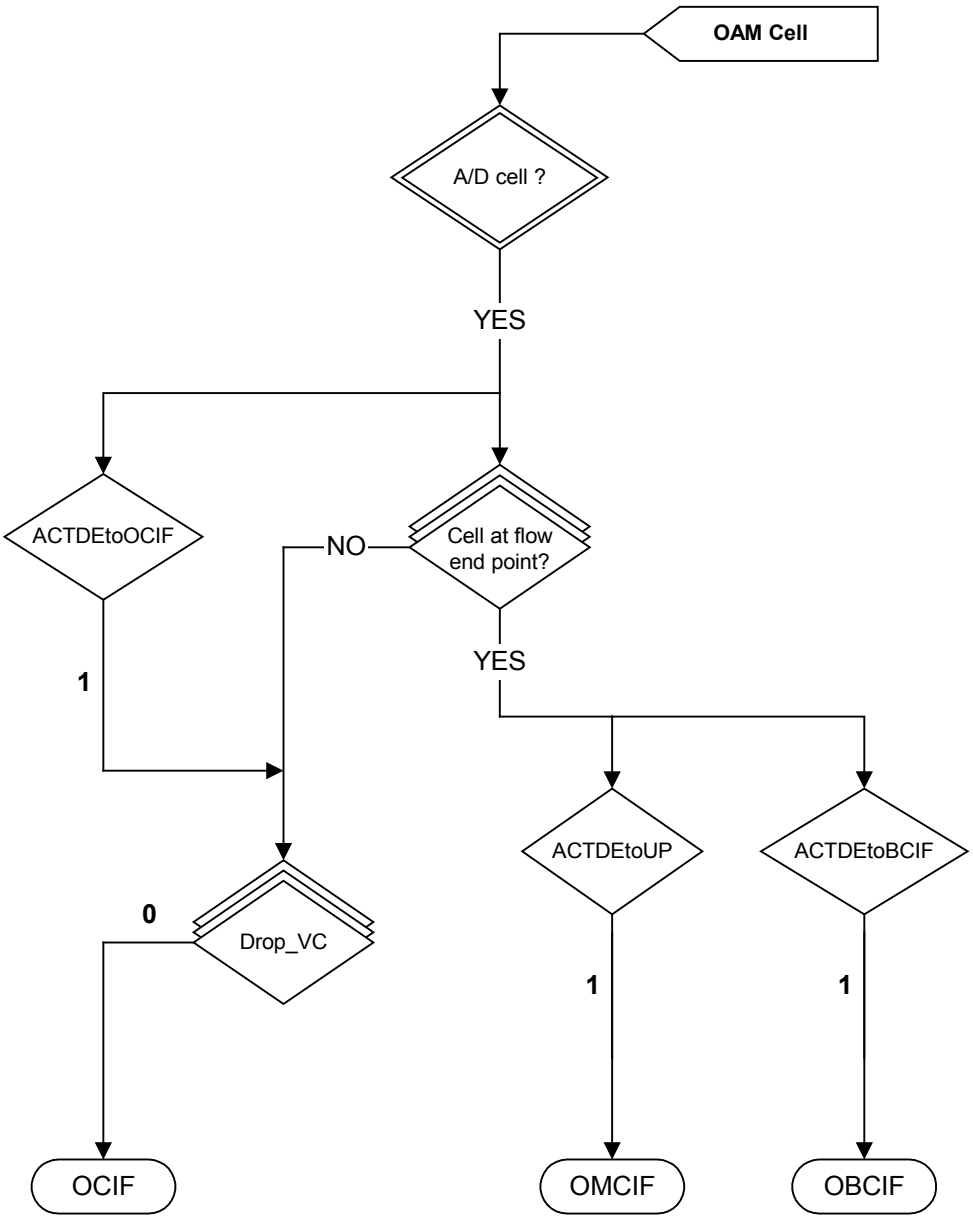


Figure 40 - APS Cell Flow

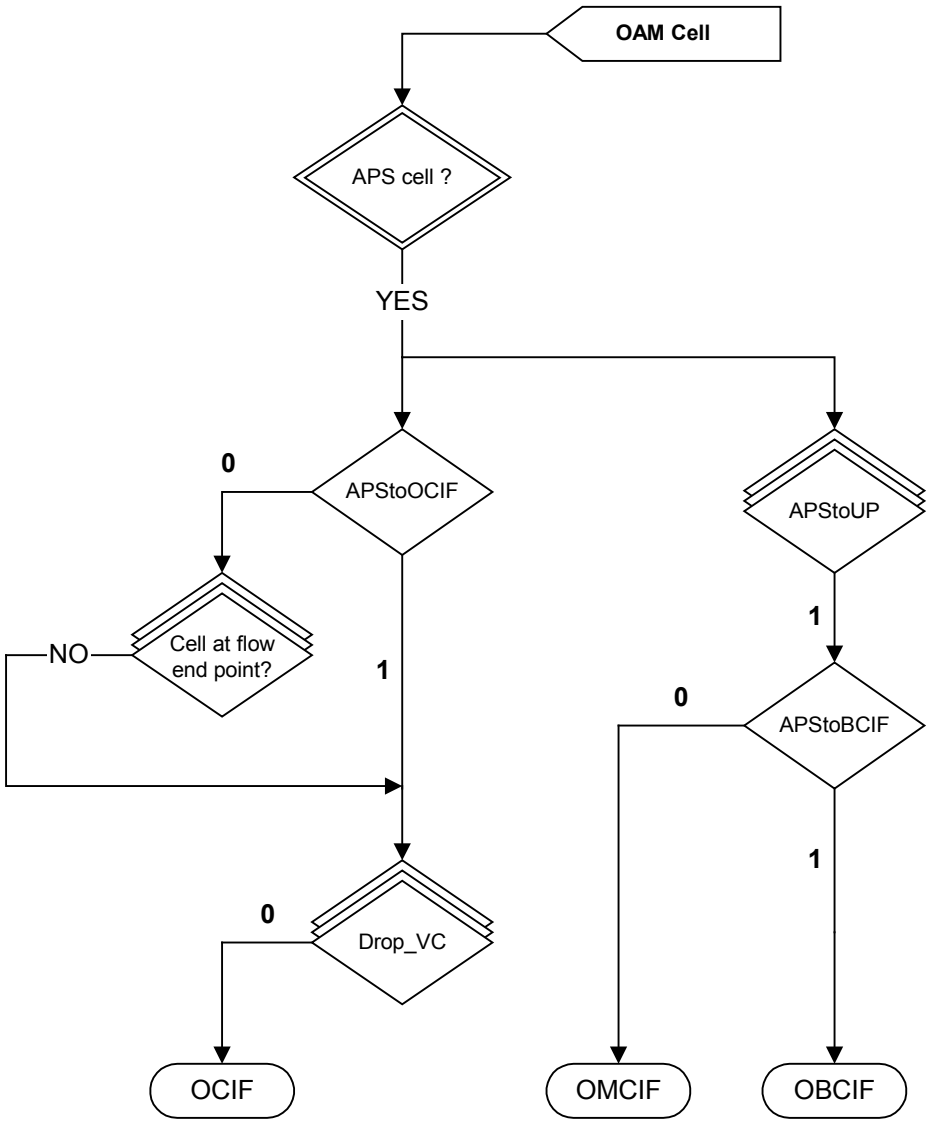


Figure 41 - System Management Cell Flow

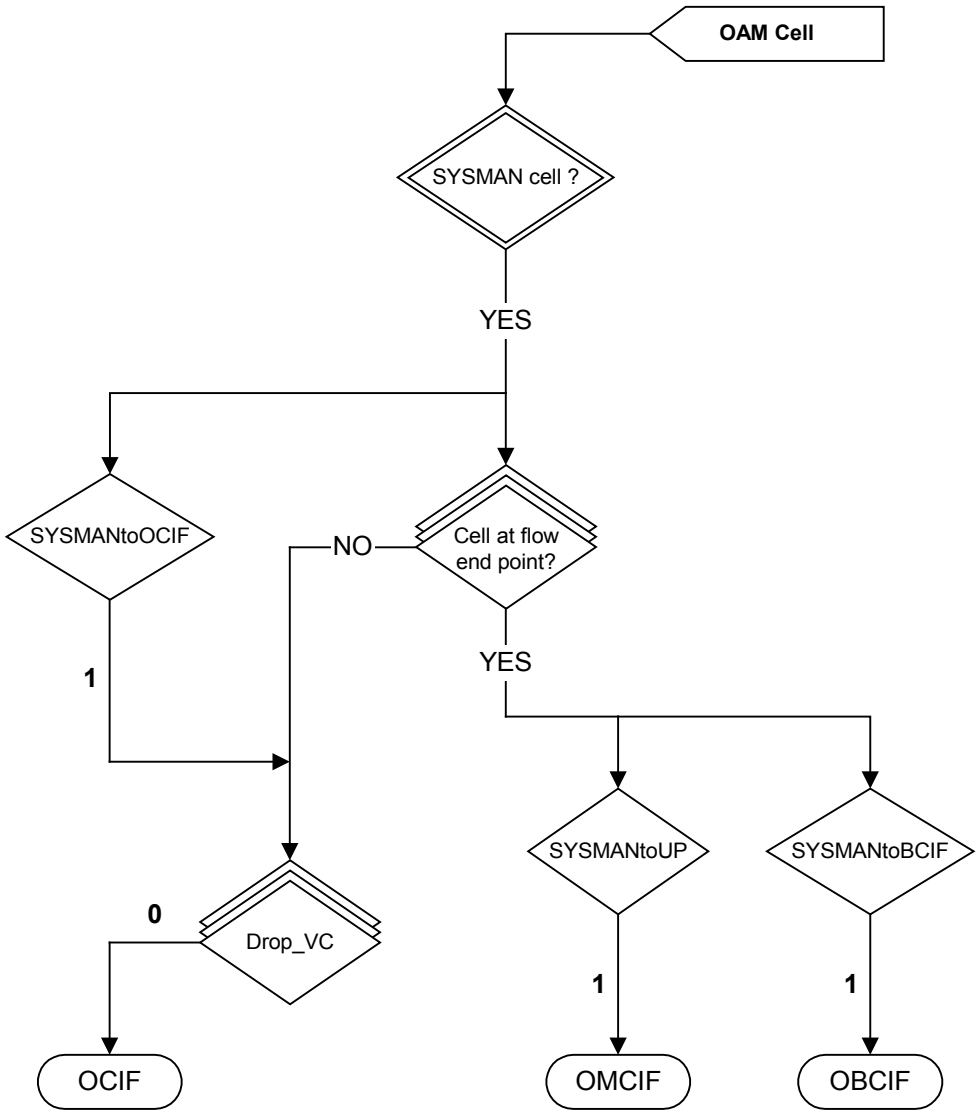
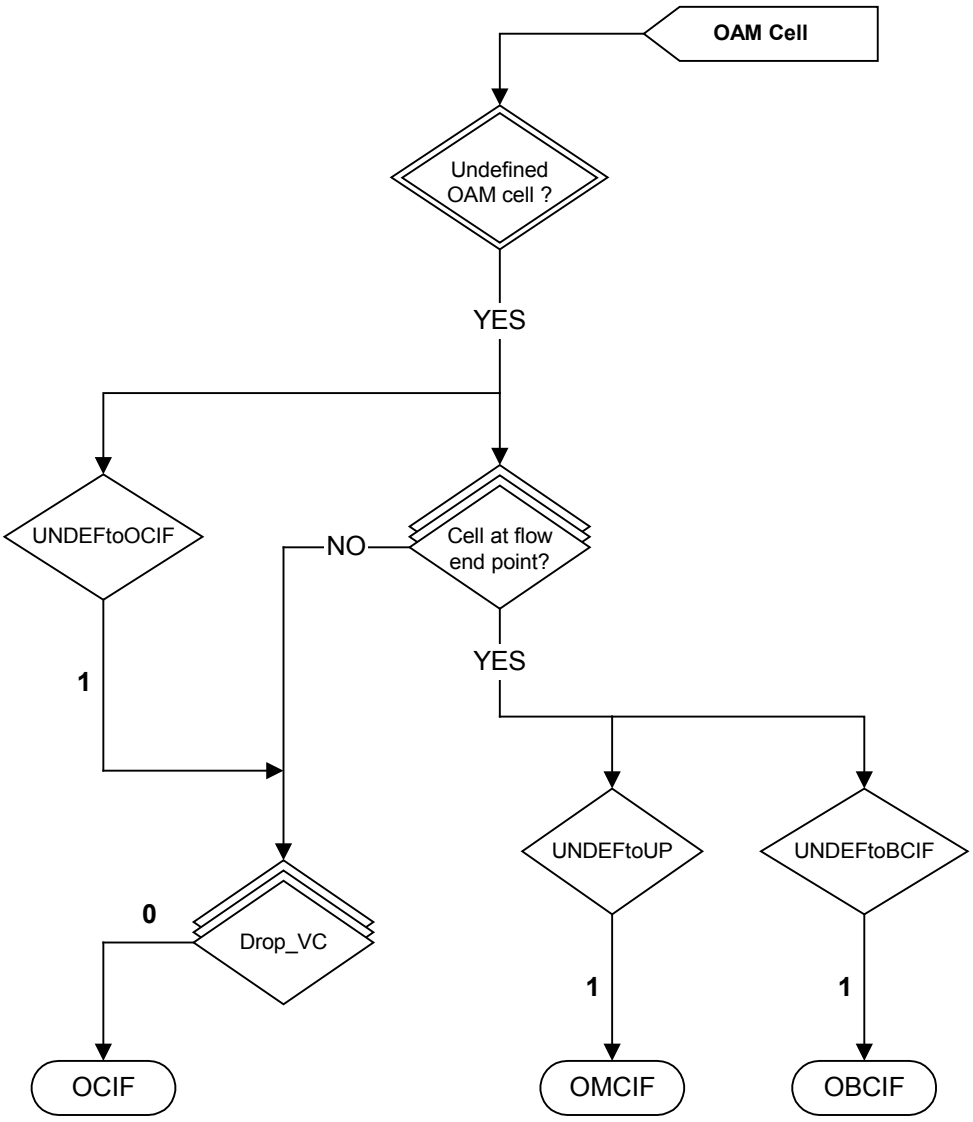
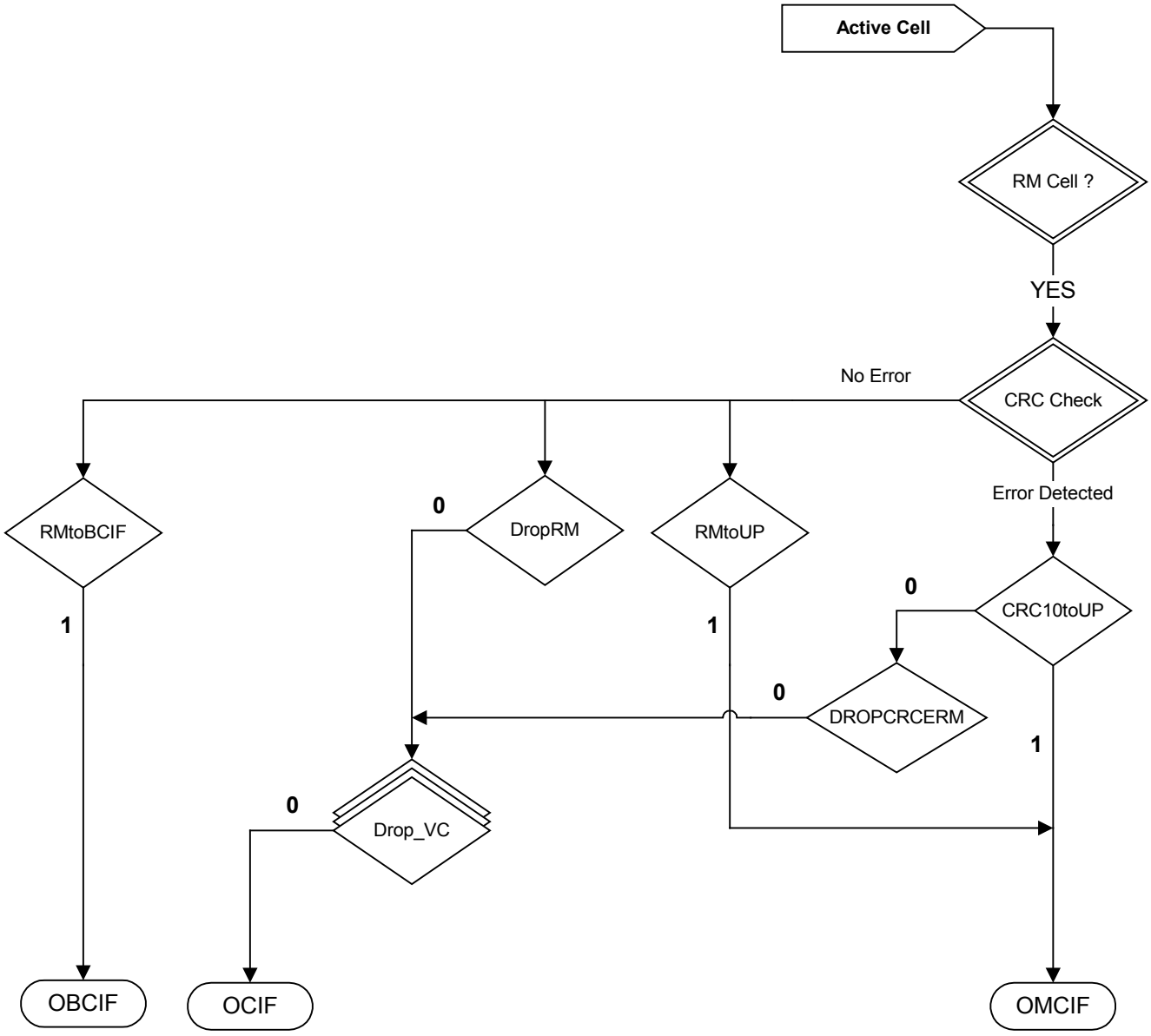


Figure 42 - OAM Cell Flow



6.12.3.3 Resource Management Cell Flow from ICIF

Figure 43 - RM Cell Flow



6.12.4 IBCIF Cell Routing

Figure 44 - Preliminary Cell Flow from the IBCIF

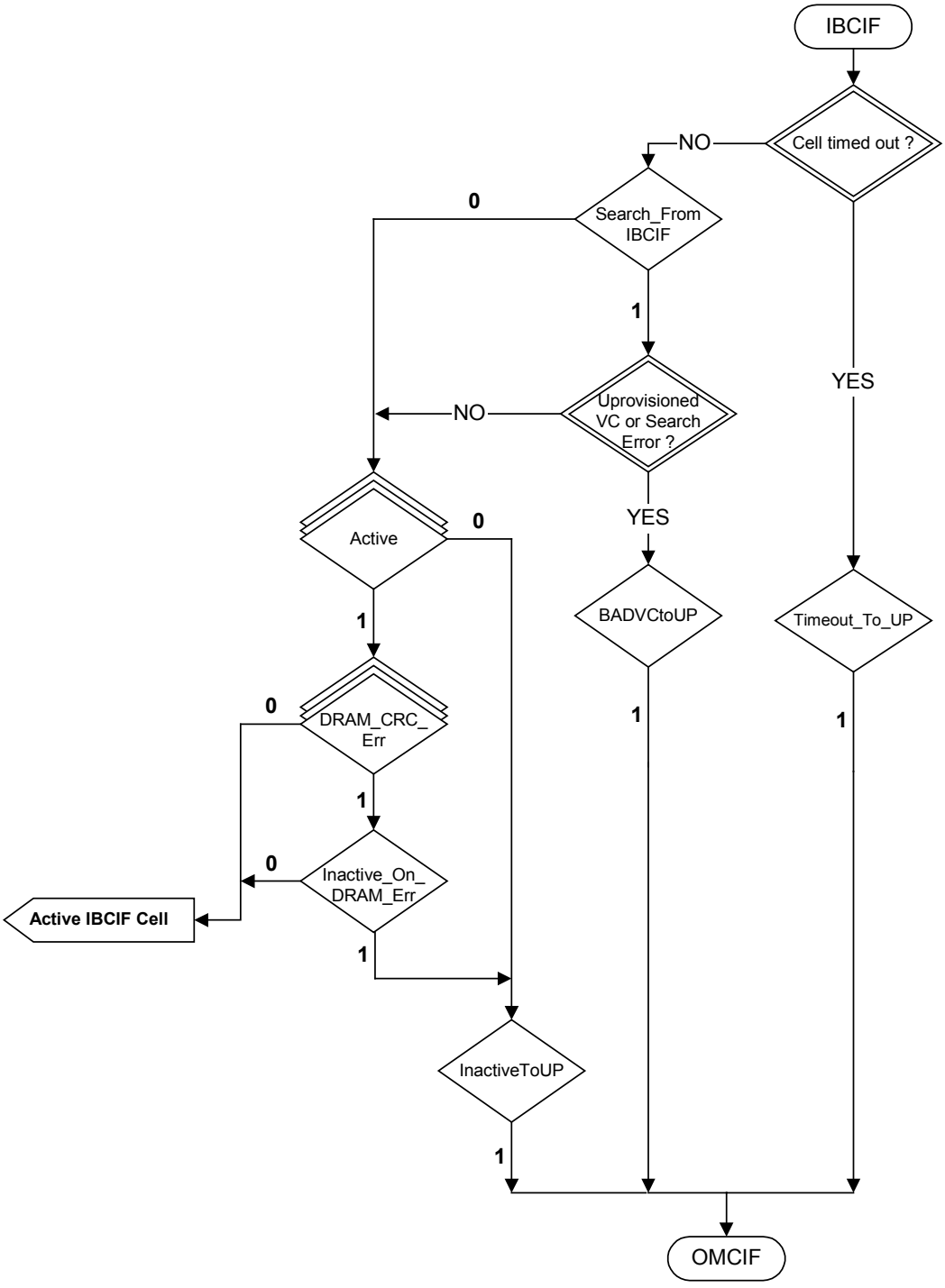


Figure 45 - User Cell Flow from the IBCIF

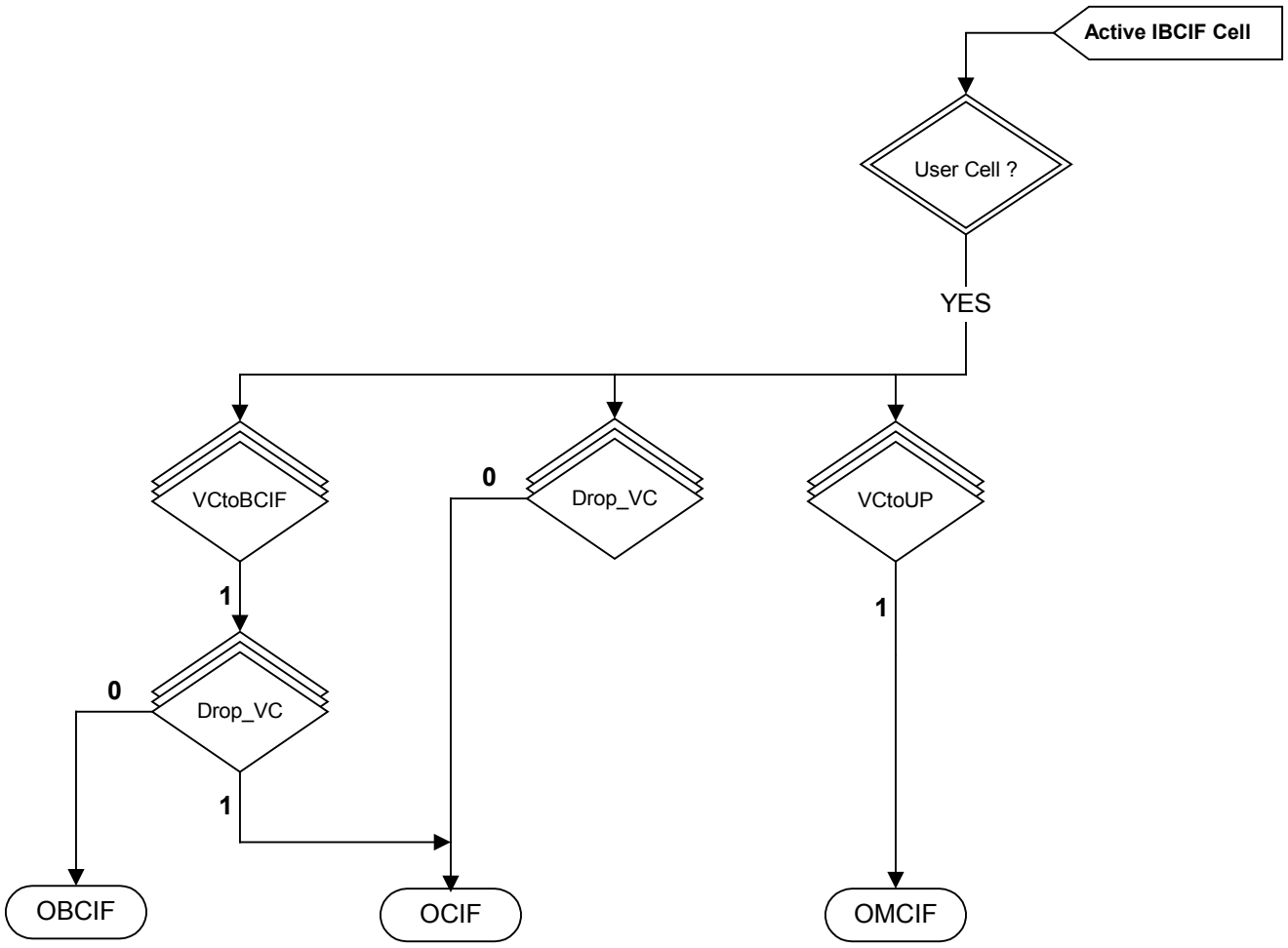
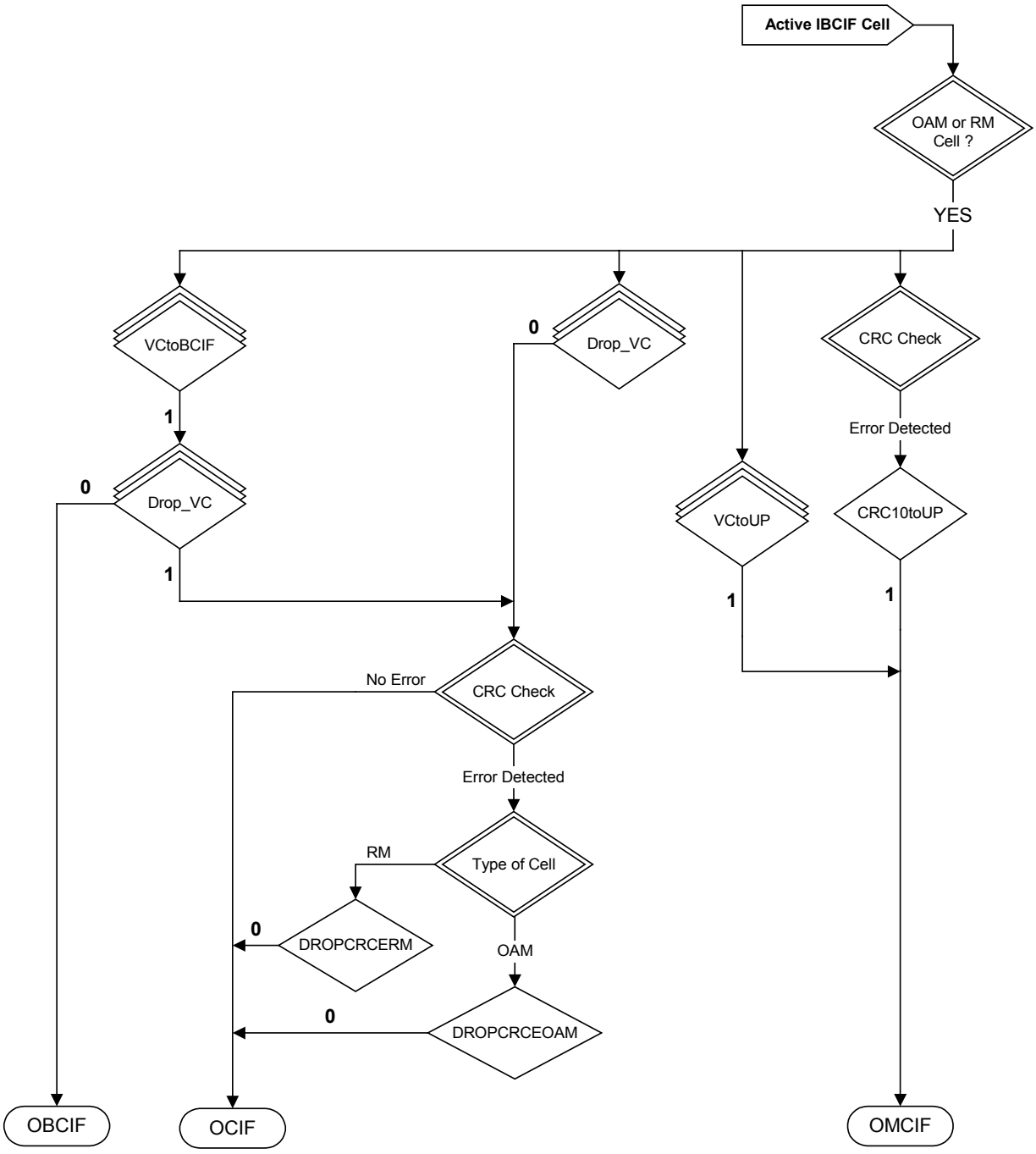


Figure 46 - OAM and RM Cell Flow from the IBCIF



7 PROGRAMMING COMPONENT INTERFACES

This section contains guides to programming the basic reading and writing interfaces to each of the ATLAS-3200 data structures shown in Figure 3, Figure 4, and Figure 5. Each data structure has a set of indirect registers that provide read and write access. The formats of the indirect registers are tailored specifically for each data structure, but in general they all include a data field, an address field, a Read/Write bit, and a BUSY bit. The general programming algorithms using the indirect register fields are similar for each data structure. The general programming algorithm for an indirect register **Write** is as follows:

1. Wait for the component to be available by checking the BUSY bit.
2. Set the address field so that the desired location of the data structure will be accessed, and set the data field to the desired value.
3. Write a logic 0 to the Read/Write bit to initiate the write cycle.

The general programming algorithm for an indirect register **Read** is as follows:

1. Wait for the component to be available by checking the BUSY bit.
2. Set the address field so that the desired location of the data structure will be accessed.
3. Write a logic 1 to the Read/Write bit to initiate the write cycle.
4. Wait for the data access to complete by checking the BUSY bit, then

For each data structure, the specific implementation of the programming algorithms, a summary of the registers involved, and example pseudo-code are provided in the sections below.

7.1 Direct Register Interfacing

All operations done by the microprocessor on the ATLAS-3200 are performed by accessing the direct microprocessor interface registers. These direct register access routines are the lowest level interface to the ATLAS-3200 and are used by all other programming routines.

7.1.1 Reading

The direct register read involves performing a single read cycle across the microprocessor interface bus. The implementation of this routine depends on the microprocessor and system hardware, so it is an application specific routine. See Section 5 for an overview of the microprocessor interface bus and see the Data Sheet [1] for bus timing information.

7.1.2 Writing

The direct register write involves performing a single write cycle across the microprocessor interface bus. The implementation of this routine depends on the microprocessor and system hardware, so it is an application specific routine. See section 5 for an overview of the microprocessor interface bus and see the Data Sheet [1] for bus timing information.

7.1.3 Example Routines

These example routines are provided as a possible implementation for a system with multiple ATLAS-3200 devices located in a memory mapped address space. There can be many possible implementations depending on the system hardware though.

7.1.3.1 regRead

This routine performs a read of the specified register of the specified ATLAS device. The specific implementation of this routine will rely entirely on the microprocessor and supporting hardware.

Inputs: regAddr : address of the register to read from

Outputs: Return value : value read from the register

Pseudocode:

```
UINT4 regRead(
    UINT2 regAddr)
{
UINT4 address
UINT4 value

address = ATLAS3200_BASE_ADDR + regAddr
value = *(address)

/* check bus errors via interrupt handling */

return value
}
```

7.1.3.2 regWrite

This routine performs a read of the specified register of the specified ATLAS device. The specific implementation of this routine will rely entirely on the microprocessor and supporting hardware.

Inputs: regAddr : address of the register to write to
value : value to write

Outputs: (none)

Pseudocode:

```
VOID regWrite(
    UINT2 regAddr,
    UINT4 value)
{
UINT4 address

address = ATLAS3200_BASE_ADDR + regAddr
*(address) = value

return
}
```

7.1.3.3 regWriteMask

This routine writes a value to a register only modifying bits which are set in the mask word. Note that since this routine performs a read, bits that clear upon read, such as interrupt indication bits, will be lost.

Inputs:

regAddr	: address of the register to write to
writeMask	: write bit mask, only bits that are set to logic 1 will be written to the bit value specified by writeValue
writeValue	: value to write to the bits that are enabled for writing

Outputs: (none)

Pseudocode:

```
VOID regWriteMask(
    UINT2 regAddr,
    UINT4 writeMask
    UINT4 writeValue)
{
UINT4 value

value = regRead(regAddr)
value = value & (~writeMask)
value = value | (writeValue & writeMask)
regWrite(regAddr, value)

return
}
```

7.1.3.4 regPollBitLow

This routine polls the specified register until either all the bits indicated in the bit mask are logic 0 or a timeout occurs. Note that since this routine performs a read, bits that clear upon read, such as interrupt indication bits, will be lost.

Inputs:

regAddr	: address of the register to containing the bit that will be polled
bitMask	: bits that are set to logic 1 in the bitMask will be polled until they all have a value of logic 0

Outputs: (none)

Pseudocode:

```
VOID regPollBitLow(
    UINT2 regAddr,
    UINT4 bitMask)
{
UINT4 value

loop TIMEOUT_NUM times
    value = regRead(regAddr)
    if (~value & bitMask) = bitMask then
        return SUCCESS
end loop
return ERROR_TIMED_OUT
}
```

7.1.3.5 regPollBitHigh

This routine polls the specified register until either all the bits indicated in the bit mask are logic 1 or a timeout occurs. Note that since this routine performs a read, bits that clear upon read, such as interrupt indication bits, will be lost.

Inputs: regAddr : address of the register to containing the bit that will be polled

bitMask : bits that are set to logic 1 in the bitMask will be polled until they all have a value of logic 1

Outputs: (none)

Pseudocode:

```
VOID regPollBitHigh(
      UINT2 regAddr,
      UINT4 bitMask)
{
UINT4 value

loop TIMEOUT_NUM times
    value = regRead(regAddr)
    if (value & bitMask) = bitMask then
        return SUCCESS
end loop
return ERROR_TIMED_OUT
}
```

7.2 External SRAM Interfacing

Access to the Search Tables and VC Linkage Table in external SRAM is provided indirectly through the registers listed in Table 51. There are a maximum of 128K Search Table Entries and 64K VC Linkage Table Records that may be accessed. The number of Tables available depends on the size of SRAM used in the specific application. The Tables are addressed by specifying an 16-bit address within either the Search partition or Linkage partition of the SRAM.

While the ATLAS-3200 is performing an SRAM access requested by the microprocessor the BUSY bit in the SRAM Access Control Register is asserted. When this bit is set the microprocessor should not initiate new SRAM access requests or read data from the SRAM Data Registers. To monitor the status of the BUSY bit the SRAM Access Control Register can be polled or the external UP_BUSY signal can be directly monitored by the microprocessor. The bits in the SRAM Access Control Register, as described in Table 52, control the indirect access cycle.

It is important that the SRAM_BUSY_EN bit in the Master Configuration and Reset (0x000) register is set to 1 so that the BUSY bit and the UP_BUSY signal are enabled. Note that the UP_BUSYB may also indicate the status of the DRAM BUSY bit. The polarity of the UP_BUSYB signal is dependent on the value of BUSYPOL in the Master Configuration and Reset Register. The bits related to SRAM configuration are described in Table 53.

This section describes the algorithms for reading and writing an SRAM entry using the indirect registers.

Table 51 - Registers for SRAM Indirect Accessing

Register	Address	Description
SRAM Access Control	0x10C	Address field and control bits. See Table 52 for details.
SRAM Data LSW	0x10D	SRAM data [31:0]
SRAM Data MSW	0x10E	SRAM data [63:32]

Table 52 - SRAM Access Control Register (0x10C)

Bit	Name	Description
31	RWB	0 : Write request initiated when this is written. 1 : Read request initiated when this is written.
30	BUSY	0 : Indicates SRAM is available for an access and values in the Data registers are stable. 1 : Indicates an SRAM Access request is pending. Do not initiate another request or read the Data registers.
17	Search/Linkage	0 : Access Search Table area of SRAM. 1 : Access Linkage Table area of SRAM.
16:0	SA[16:0]	SRAM Address in either the Search or Linkage areas of SRAM.

Table 53 - Register Bits for SRAM Config. and Interrupts

Bit Name	Register	Description
SRAM_BUSY_EN	Master Configuration and Reset (0x000)	Enables the UP_BUSY signal and BUSY bit in the SRAM Access Control Register (0x10C). ***IMPORTANT:*** The default value of this bit is 0, however use of the BUSY signal is required to ensure correct SRAM accesses so this enable bit should always be set to 1.
SRAM_Even_Parity	Cell Processor Configuration (0x100)	0 : Even parity is generated and checked during SRAM accesses. 1 : Odd parity is generated and checked during SRAM accesses.

Bit Name	Register	Description
SPRTYI[7:0]	Master Interrupt Status #2 (0x003)	Interrupt status flags indicating parity errors over SDAT[63:0]. See Datasheet [1] for details.
SPRTYE[7:0]	Master Interrupt Enable #2 (0x005)	Enable interrupts generated by the corresponding interrupt status flags.

7.2.1 Reading External SRAM Entries

To perform a read, execute the following steps:

1. Check that the BUSY bit in the SRAM Access Control Register is deasserted. Do not proceed if the BUSY bit is asserted.

2. Write to the SRAM Access Control Register with the following settings:

    - RWB bit set to logic 1
    - Search/Linkage bit set to logic 0 if reading a Search Table Entry or set to logic 1 if reading a VC Linkage Table.
    - SA[16:0] set to the address of the entry to be read

    This will initiate the SRAM write cycle and assert the BUSY bit.

3. Wait until the BUSY bit is deasserted. This indicates that the SRAM access is complete.

4. Read the data from the SRAM Data Registers.

7.2.2 Writing External SRAM Entries

To perform a write, execute the following steps:

1. Check that the BUSY bit in the Access Control Register is deasserted. Do not proceed if the BUSY bit is asserted.

2. Write data to the SRAM Data MSW and SRAM Data LSW registers as required.

3. Write to the SRAM Access Control Register with the following settings:

- RWB bit set to logic 0
- Search/Linkage bit set to logic 0 if writing a Search Table Entry or set to logic 1 if writing a VC Linkage Table.
- SA[16:0] set to the address of the entry to be written

This will initiate the SRAM write cycle and assert the BUSY bit. When the BUSY bit is deasserted the SRAM access is complete.

7.2.3 Diagnostic Testing

Writing and reading a series of test patterns can test the external SRAM and its interface to the ATLAS-3200. There are many possible test procedures that can be used but a test algorithm based on "Fault Modeling and Test Algorithm Development for Static Random Access Memories"[4] is performed as follows:

1. Write "pattern" to all locations with an incrementing address.
2. Read "pattern" and write inverse "pattern" to all locations with an incrementing address.
3. Read inverse "pattern" and write "pattern" to all locations with an incrementing address.
4. Read "pattern" and write inverse "pattern" to all locations with a decrementing address.
5. Read inverse "pattern" and write "pattern" to all locations with a decrementing address.

Steps 1 to 5 are repeated using the "pattern" values of 0x00, 0x55 and 0xCD repeated over the eight byte width of the SRAM bus. The last pattern verifies the parity bits.

7.2.4 Example Routines

7.2.4.1 Definitions

```
typedef struct {
    UNIT4 msw,
    UINT4 lsw
} sSRAM_DATA

typedef enum {
```

```
    SRAM_SEARCH = 0,
    SRAM_LINKAGE
} SRAM_PARTITION_TYPE
```

7.2.4.2 sramRead

This routine reads a row from the external SRAM. Polling of the BUSY bit in the SRAM Access Control Register is used. Modifications could be made to use the external UP_BUSYB signal. The specific implementation of this routine will rely entirely on the microprocessor and supporting hardware.

Inputs:	address	: SRAM address in either the Linkage or Search Table partition
	sramPartition	: Specifies access to either the Linkage or Search table SRAM partitions
Outputs:	sramData	Pointer to SRAM data structure that will contain the 64-bit value that was read

Pseudocode:

```
VOID sramRead(
    UINT2 address,
    SRAM_PARTITION_TYPE sramPartition,
    sSRAM_DATA *sramData)
{
UINT4 controlWord

/*ensure BUSY bit is low*/
regPollBitLow(REG_SRAM_ACCESS_CTRL, BTMSK_SRAM_BUSY)

/*write to control register to initiate transfer*/
controlWord = (BTMSK_SRAM_RWB | (BTMSK_SRAM_SA & address))
if (sramPartition = SRAM_LINKAGE ) then
     controlWord = controlWord | BTMSK_SRAM_SEARCH_LINKAGE
end if
regWrite(REG_SRAM_CONTROL, controlWord)

/*wait for transfer to complete*/
regPollBitLow(REG_SRAM_CONTROL, BTMSK_SRAM_BUSY)

/*read data*/
sramData->msw = regRead(REG_SRAM_DATA_MSW)
```

```
sramData->lsw = regRead(REG_SRAM_DATA_LSW)

/*handle SRAM parity errors though interrupt handler*/

return
}
```

7.2.4.3 sramWrite

This routine writes a row to the external SRAM. Polling of the BUSY bit in the SRAM Access Control Register is used. Modifications could be made to use the external UP_BUSYB signal. The specific implementation of this routine will rely entirely on the microprocessor and supporting hardware.

Inputs:	address	: SRAM address in either the Linkage or Search Table partition
	sramPartition	: Specifies access to either the Linkage or Search table SRAM partitions
	sramData	Pointer to SRAM data structure that will contain the 64-bit value that will be written

Outputs: (none)

Pseudocode:

```
VOID sramWrite(
    UINT4 address,
    SRAM_PARTITION_TYPE sramPartition,
    sSRAM_DATA *sramData)
{
UINT4 controlWord

/* ensure BUSY bit is low */
regPollBitLow(REG_SRAM_CONTROL, BITPOS_SRAM_CONTROL_BUSY)

/* write data */
regWrite(REG_SRAM_DATA_MSW, &sramData->msw)
regWrite(REG_SRAM_DATA_LSW, &sramData->msw)

/* write to control register to initiate transfer */
controlWord = (BTMSK_SRAM_SA & address)
if (sramPartition = SRAM_LINKAGE) then
    controlWord = controlWord | BTMSK_SRAM_SEARCH_LINKAGE
```

```
end if
regWrite(REG_SRAM_CONTROL, controlWord)

/* handle SRAM parity errors though interrupt handler */

return SUCCESS
}
```

7.2.4.4 sramTest

This function performs a diagnostic test of the SRAM by reading and writing a series of test patterns. Depending on the depth of the SRAM, this test may take several seconds.

The global constant, SRAM_MAX_ADDRESS, determines the depth of RAM tested. Note that the test overwrites the contents of SRAM.

Inputs: (none)

Outputs:

faultAddress	: address of the first SRAM fault location that was found
faultPartition	: partition that the first SRAM fault was located in

Pseudocode:

```
UINT1 sramTest(
    UINT4 *faultAddress
    SRAM_PARTITION_TYPE *faultPartition)
{
UINT1 pattern[MAXPATTERN]={ 0x00, 0x55, 0xCD }
UINT4 patternWord
sSRAM_DATA sramPattern
sSRAM_DATA sramInvPattern
UINT4 address

for patternIndex = 0 to MAXPATTERN - 1

    patternWord = (pattern[patternIndex] << 24) |
                  (pattern[patternIndex] << 16) |
                  (pattern[patternIndex] << 8) |
                   pattern[patternIndex]

    sramPattern.lsw = patternWord
```

```
    sramPattern.msw = patternWord

    sramInvPattern.lsw = ~patternWord
    sramInvPattern.msw = ~patternWord

    /* Write "pattern" with an incrementing address */
    for address = 0 to SRAM_MAX_ADDRESS
        sramWrite(address, SRAM_LINKAGE, sramPattern)
        sramWrite(address, SRAM_SEARCH, sramPattern)
    next address

    /* Read "pattern" and write inverse "pattern" with an incrementing
       address */
    for address = 0 to SRAM_MAX_ADDRESS
        /* test Linkage partition */
        if sramRead( address, SRAM_LINKAGE, sramPattern) != patternWord
           then
            faultAddress = address
            faultPartition = SRAM_LINKAGE
            return FAILURE
        end if
        sramWrite( address, SRAM_LINKAGE, sramInvPattern )

        /* test Search partition */
        if sramRead( address, SRAM_SEARCH, sramPattern ) != patternWord
           then
            faultAddress = address
            faultPartition = SRAM_SEARCH
            return FAILURE
        end if
        sramWrite( address, SRAM_SEARCH, sramInvPattern )
    next address

    /* Read inverse "pattern" and write "pattern" to all locations with
       incrementing addresses */

    /* Read "pattern" and write inverse "pattern" to all locations with
       decrementing addresses */

    /* Read inverse "pattern" and write "pattern" to all locations with
       decrementing addresses */
next patternIndex

return SUCCESS
}
```

7.3 VC Table Entry Interfacing

Access to the VC Table Records in the VC RAM is provided indirectly through the registers listed in Table 54. The VC RAM has 64K VC Table Records that are addressed by specifying the a 16-bit Virtual Connection Record Address (VCRA). The bits in the VC Table Access Control Register, as described in Table 55, control the indirect access cycle.

This section describes the algorithms for reading and writing a VC Table Record using the indirect registers.

Table 54 - Registers for VC Table Entry Interfacing

Register	Address	Description
VC Table Access Control	0x111	Address field and control bits
VC Table Write Enable 1	0x112	Write enables for individual fields in the VC Table Record
VC Table Write Enable 2	0x113	Write enables for individual fields in the VC Table Record
VC Table Data Row 0, Word 0 (LSW) (RAM Data [31:0])	0x114	Data registers corresponding to the rows of a VC Table Record. Write to these registers before initiating a VC Table Record write operation, or read them after completing a VC Table Record read operation.
VC Table Data Row 0, Word 1 (RAM Data [63:32])	0x115	
VC Table Data Row 0, Word 2 (RAM Data [95:64])	0x116	
VC Table Data Row 0, Word 3 (MSW) (RAM Data [127:96])	0x117	
(VC Table Data Row registers continue for Rows 2 to 6) …		

Table 55 - VC Table Access Control Register (0x111)

Bit	Name	Description
31	RWB	0 : Write request initiated when this is written. 1 : Read request initiated when this is written.

Bit	Name	Description
30	BUSY	0 : Indicates SRAM is available for an access and values in the Data registers are stable. 1 : Indicates an SRAM Access request is pending. Do not initiate another request or read the Data registers.
29	CC_ClearOnRd	0 : Cell Count fields are unchanged after a read. 1 : After a read access, the Cell Count fields of the VC Table are written to all 0's and the VC Table's CRC is updated.
28	AC_ClearOnRd	0 : Alternate Cell Count fields are unchanged after a read. 1 : After a read access, the Alternate Cell Count fields of the VC Table are written to all 0's and the VC Table's CRC is updated.
27	NCC_ClearOnRd	0 : Non-compliant Cell Count fields are unchanged after a read. 1 : After a read access, the Non-compliant Cell Count fields of the VC Table are written to all 0's and the VC Table's CRC is updated.
26:24	Unused	Always set to logic 0 when writing. Read value is undefined.
23	DRAM_CRC_ERR	0 : No CRC error occurred during the last VC Table transfer. 1 : Indicates CRC of the VC Table that was transferred was incorrect. Will be set after a read or write access if an error was detected.
22:17	Unused	Always set to logic 0 when writing. Read value is undefined.
16	Reserved	Always set to logic 0 when writing. Read value is undefined.
15:0	VCRA[15:0]	Virtual Connection Record Address specifying which VC Table to access

7.3.1 Reading VC Table Records

The procedure to read a VC Table Record from a specified VCRA is as follows:

1. Poll the BUSY bit in the VC Table Access Control Register until its value is 0. This indicates that the VC RAM is ready to perform an access.

2. Write to the VC Table Access Control Register with the following settings:

   - RWB = 1,
   - CC_CLRONRD, AC_CLRONRD, NCC_CLRONRD set depending on the application for optional clearing of the Cell Count, Alternate Cell Count, and Non-Compliant Cell Count fields
   - VCRA[15:0] set to address of the VC Table Record that will be read

   This will initiate the VC RAM read cycle and assert the BUSY bit.

3. Poll the BUSY bit in the VC Table Access Control Register until its value is 0. This indicates that the specified VC Table Record has been read and the data in the VC Table Data Registers has been updated.

4. Check that the DRAM_CRC_ERR bit in the VC Table Access Control Register is set to logic 0. If the DRAM_CRC_ERR is logic 1 then an error during the memory access and the data in the VC Table Data Registers may be invalid.

5. Read the VC Table Data Registers as desired.

7.3.2 Writing VC Table Records

The procedure to write a VC Table Record to a specified VCRA is as follows:

1. Poll the BUSY bit in the VC Table Access Control Register until its value is 0. This indicates that the VC RAM is ready to perform an access.

2. Write the desired values to the VC Table Data Registers. Only the fields that will be not be write protected need to be written. Write protection is controlled on per-field basis in the VC Table Write Enable Registers.

3. Write the desired bits in the VC Table Write Enable Registers to select which fields will be written to the VC Table Record. Fields that have their enable bit set to logic 1 will be written with the value in the VC Table Data Register and those with their enable bit set to logic 0 will remain unchanged.

NOTE: *Ensure that the Reserved [20] bit in VC Table Write Enable 1 Register is always set to logic 0.*

4. Write to the VC Table Access Control Register with the following settings:

- RWB = 0,
- VCRA[15:0] set to address of the VC Table Record that will be written

This will initiate the VC RAM write cycle and assert the BUSY bit. When the BUSY bit is deasserted the VC Table Record has been written.

7.3.3 Example Routines

7.3.3.1 vcRecordTableWrite

This routine writes a VC Table Record to the specified address. The information in the VC Table Record data structure is written to the indirect data registers.

Inputs:

vcra	: VC Record Address of the VC Table Record that will be written
vcRecordTable	: pointer to a VC Table Record structure that will be written to memory

Outputs: (none)

Pseudocode:

```
UINT1 vcRecordTableWrite(
    UINT2               vcra,
    sVC_RECORD_TABLE    *vcRecordTable)
{
UINT4  vcRecordTableRaw[4][7]
UINT4  controlWord
UINT1  row
UINT1  word

/*ensure BUSY bit is low*/
regPollBitLow(REG_VC_TABLE_ACCESS_CTL, BTMSK_VC_ACCESS_CTL_BUSY)

/* serialize the fields in the VC Table Record data structure  */
/* to the format that is used in the actual VC Table Record in */
/* the ATLAS-3200 */
serializeVcRecordTable(vcRecordTable, vcRecordTableRaw)

/* load the data registers */
```

```
for row = 0 to 6
    for word = 0 to 3
        regWrite(REG_VC_TABLE_DATA + (row*4) + word,
                 vcRecordTableRaw[word,row])
    next word
next row

/* set the write enables */
regWrite(REG_VC_TABLE_WRITE_EN1, /*desired write enable value*/)
regWrite(REG_VC_TABLE_WRITE_EN2, /*desired write enable value*/)

/* write to control register to initiate the write */
controlWord = (vcra & BTMSK_VC_ACCESS_CTL_VCRA) /* RWB will be 0 */
regWrite(REG_VC_TABLE_ACCESS_CTL, controlWord)

/* check if a DRAM Error occurred */
controlWord = regRead(REG_VC_TABLE_ACCESS_CTL)
if (controlWord && BTMSK_VC_ACCESS_CTL_DRAMERR =
    BTMSK_VC_ACCESS_CTL_DRAMERR) then
    return FAILURE
end if

return SUCCESS
}
```

7.3.3.2 vcRecordTableRead

This routine reads a VC Table Record from the specified address. A VC Table Record data structure is filled with the data read from the indirect data registers.

Inputs:

vcra	: VC Record Address of the VC Table Record that will be read
vcRecordTable	: pointer to a VC Table Record structure that will be filled with the data that was read

Outputs: (none)

Pseudocode:

```
UINT1 vcRecordTableRead(
    UINT2               vcra,
    sVC_RECORD_TABLE    *vcRecordTable)
{
```

```
UINT4  vcRecordTableRaw[4][7]
UINT4  controlWord
UINT1  row
UINT1  word

/* ensure BUSY bit is low */
regPollBitLow(REG_VC_TABLE_ACCESS_CTL, BTMSK_VC_ACCESS_CTL_BUSY)

/* write to control register to initiate the read */
controlWord = (vcra & BTMSK_VC_ACCESS_CTL_VCRA) |
              (BTMSK_VC_ACCESS_CTL_RWB) |
               /* optionally set clear on read bits if desired */
regWrite(REG_VC_TABLE_ACCESS_CTL, controlWord)

/* wait for transfer to complete */
regPollBitLow(REG_VC_TABLE_ACCESS_CTL, BTMSK_VC_ACCESS_CTL_BUSY)

/* check if DRAM Error occurred */
controlWord = regRead(REG_VC_TABLE_ACCESS_CTL)
if (controlWord && BTMSK_VC_ACCESS_CTL_DRAMERR =
    BTMSK_VC_ACCESS_CTL_DRAMERR) then
    return FAILURE
end if

/* load the raw VC Table Record array with data registers values */
for row = 0 to 6
    for word = 0 to 3
        vcRecordTableRaw[word,row]) = regRead(REG_VC_TABLE_DATA +
                                              (row*4) + word)
    next word
next row

/* convert the raw VC Table Record data into field values for the */
/* microprocessor's VC Table Record data structure                */
convertVcRecordTable(vcRecordTableRaw, vcRecordTable)

return SUCCESS
}
```

7.4 SDQ Entry Interfacing

Access to the three Scaleable Data Queues (SDQs) is provided indirectly through the registers listed in Table 56. Each SDQ has 48 FIFO configuration entries that correspond to each of the 48 PHYs. The entries are addressed by specifying the PHY ID of the entry that will be accessed. The bits in the Indirect Address Register, as described in Table 57, control the indirect access cycle.

This section describes the algorithms for reading and writing a FIFO configuration entry using the indirect access registers. The algorithms apply to all three SDQs.

When reconfiguring the SDQ while traffic is passing through, all FIFOs that are being affected by the reconfiguration should be disabled and flushed, and held that way until the reconfiguration is complete. FIFOs unaffected by the reconfiguration will continue to carry traffic normally.

Table 56 - Registers for SDQ Entry Indirect Programming

Input SDQ Registers	Output SDQ Registers	Bypass SDQ Registers	Description
Input SDQ Indirect Address (0x244)	Output SDQ Indirect Address (0x2A4)	Bypass SDQ Indirect Address (0x2C4)	Indirect access control bits and physical ID address. The physical ID addresses the per-PHY FIFO data accessed by the other Indirect registers.
Input SDQ Indirect Configuration (0x245)	Output SDQ Indirect Configuration (0x2A5)	Bypass SDQ Indirect Configuration (0x2C5)	FIFO configuration data to write or read
Input SDQ Indirect Cells and Packets Count (0x246)	Output SDQ Indirect Cells and Packets Count (0x2A6)	Bypass SDQ Indirect Cells and Packets Count (0x2C6)	Read-only count of the number of packets or ATM cells in the specified FIFO.

Table 57 - SDQ Indirect Address Regs (0X244,0X2A4,0X2C4)

Bit	Name	Description
31:16	Unused	Always set to logic 0 when writing. Read value is undefined.
15	BUSY	0 : Indicates PHY Mapping Table is available for an access and value in the Data register is stable. 1 : Indicates a PHY Mapping Table Access request is pending. Do not initiate another request or read the Data register.

Bit	Name	Description
14	RWB	0 : Write request initiated when this is written. 1 : Read request initiated when this is written.
13	FLUSH	0 : Writes to this register will not affect FIFO data. 1 : Writes to this register will to cause the FIFO addressed by the PHY_ID[5:0] to discard all cells or packets in it. The FIFO will then be empty. Use this before re-configuring a FIFO.
12	EMPTY	0 : Indicates the FIFO addressed by the PHY_ID[5:0] field contains at least one cell or packet. 1 : Indicates the FIFO addressed by the PHY_ID[5:0] field is empty.
11:6	Unused	Always set to logic 0 when writing. Read value is undefined.
5:0	PHY_ID[5:0]	PHY ID that addresses the entry in the PHY Mapping Table that will be accessed.

7.4.1 Reading SDQ Entries

The procedure to read the entry for a given physical connection from an SDQ is as follows:

1. Poll the BUSY bit in the SDQ Indirect Address Register until its value is 0. This indicates that the SDQ is ready to perform an access.

2. Write to the SDQ Indirect Address Register with the following settings:

    - RWB set to logic 1
    - FLUSH set to logic 0
    - PHYID[5:0] set to the desired physical connection index

    This will initiate the SDQ read cycle and assert the BUSY bit.

3. Poll the BUSY bit in the SDQ Indirect Address Register until its value is 0. This indicates that the SDQ is has completed the read cycle and the data in the indirect registers have been updated.

4. Read the SDQ Indirect Configuration Register, SDQ Indirect Buffer and Data Available Thresholds Register, and Input SDQ Indirect Cell/Packet Count Register as desired.

7.4.2 Writing SDQ Entries

The procedure to write an entry for a given physical connection to an SDQ is as follows:

1. Poll the BUSY bit in the SDQ Indirect Address Register until its value is 0. This indicates that the SDQ is ready to perform an access.

2. Determine the current FIFO configuration settings. If the microprocessor does not maintain a copy of this information, obtain it by reading the current SDQ entry as described in Section 7.4.1 Reading SDQ Entries.

3. Disable the current FIFO by writing the to the SDQ Indirect Configuration Register with the following settings.

    - FIFO_PTR, FIFO_SIZE, FIFO_TYPE set to the current settings
    - FIFO_ENBL bit set to logic 0. Setting the Enable bit to 0 will ensure that no cells arrive in the FIFO while it is being configured.

    Then write to the SDQ Indirect Address Register with the following settings:

    - RWB bit set to logic 0
    - FLUSH bit set to logic 1
    - PHYID[5:0] set to the desired physical connection index

4. Poll the BUSY bit in the SDQ Indirect Address Register until its value is 0.

5. Configure and enable the new FIFO settings by writing to the SDQ Indirect Configuration Register with the following settings:

    - FIFO_PTR, FIFO_SIZE, FIFO_TYPE set to the desired settings for the new FIFO.
    - FIFO_ENBL bit set to logic 1

    Then write to the SDQ Indirect Address Register with the following settings:

    - RWB bit set to logic 0

- FLUSH bit set to logic 0
- PHYID[5:0] set to the desired physical connection index

This will write the Enable bit from the Configuration Register and clear the Flush bit so that the FIFO will be enabled and ready to receive cells. When the BUSY bit is deasserted the SDQ access is complete.

7.5 PM Table Record Interfacing

Access to the Performance Management (PM) Tables in the two PM Table Banks is provided indirectly through the registers listed in Table 58. Each PM Table Bank has 256 PM Table Records that are addressed by specifying the index of the PM Table Record and by specifying the Bank in which it is located. The bits in the Control Register, as described in Table 59, control the indirect access cycle.

This section describes the algorithms for reading and writing a PM Table Record using the indirect access registers.

Table 58 - Registers for PM Table Indirect Accessing

Register	Address	Description
Performance Management RAM Record Address, Word Select and Access Control	0x170	Address field and control bits
Performance Management RAM Row 0 Word 0 (LSW)	0x171	Data registers corresponding to the rows of a PM Table Record. Write to these registers before initiating a PM Table Write, or read them after completing a PM Table Read.
Performance Management RAM Row 0 Word 1	0x172	
Performance Management RAM Row 0 Word 2 (MSW)	0x173	
… (continues for Rows 1 to 6) …		
Performance Management RAM Row 7 Word 0 (LSW)	0x186	
Performance Management RAM Row 7 Word 1	0x187	
Performance Management RAM Row 7 Word 2 (MSW)	0x189	

Table 59 - PM Word Select and Access Control Reg (0x170)

Bit	Name	Description
31:25	Unused	Always set to logic 0 when writing. Read value is undefined.
24	PM Bank	0 : Access to PM Table Record in PM Bank 1 1 : Access to PM Table Record in PM Bank 2
23:16	PM Addr [7:0]	Index of PM Table Record that will be accessed within the selected bank
15	RWB	0 : Write request initiated when this is written. 1 : Read request initiated when this is written.
14	BUSY	0 : Indicates PM RAM available for access and values in the Data registers are stable. 1 : PM RAM access request is pending. Do not initiate another request or read the Data registers.
13:9	ClrOnRd_Row[7:3]	Bitmap specifying which rows to automatically clear following a Read access. Rows 0,1, and 2 cannot be automatically cleared following a read.
8:1	Wr_PM_Row[7:0]	Bitmap specifying a write mask for Rows 7 to 0. Only Rows with bit set to 1 will be altered during a read. The Config field in Row 0 is independently write protected by the Wr_PM_Config bit.
0	Wr_PM_Config	0 : PM Configuration & Status field in Row 0 will be left unchanged after a Write access. 1 : PM Configuration & Status field will be written during Write access. This bit can be used to re-configure the PM Table Record without changing any of the current counter values.

7.5.1 Reading PM Table Records

The general procedure to use the indirect registers to read a PM Table Record is as follows:

1. Poll the BUSY bit in the PM Control Register (0x170) until its value is 0. This indicates that the PM RAM is ready to perform an access.

2. Write to the PM Control Register (0x170) with the following settings:

    - RWB set to logic 1
    - ClrOnRd_Row[7:3] set so that the desired rows will be cleared
    - PM Bank and PM Addr[7:0] set to the location of the PM Table Record that will be read

    This will initiate the PM RAM read cycle and assert the BUSY bit.

3. Poll the BUSY bit in the PM Control Register (0x170) until its value is 0. This indicates that the specified PM Table Record has been read and the data in the PM RAM Row Registers has been updated.

4. Read the PM RAM Row Registers as desired.

7.5.2 Writing PM Table Records

The general procedure to use the indirect registers to write a PM Table Record is as follows:

1. Poll the BUSY bit in the PM Control Register (0x170) until its value is 0. This indicates that the PM RAM is ready to perform an access.

2. Write the desired values to the PM RAM Row Registers. Only the locations that will be not be write protected need to be written. The write protection is controlled on per-row basis except for the PM Configuration & Status field in Row 0 which has the independent write mask bit.

3. Write to the PM Control Register (0x170) with the following settings:

    - RWB set to logic 0
    - Write mask fields set to logic 1 for fields that are desired to be written to. Note that the Wr_PM_Config bit supercedes the value of

Wr_PM_Row[0], so the PM Configuration & Status field can be programmed independently.

- PM Bank and PM Addr[7:0] set to the location of the PM Table Record that will be written to

This will initiate the PM RAM write cycle and assert the BUSY bit. When the BUSY bit is deasserted the PM Table Record has been written.

7.6 PHY ID Mapping Table Interfacing

Access to the entries in the two PHY ID Mapping Tables is provided indirectly through the registers listed in Table 60. Each PHY ID Mapping Table has 48 entries that correspond to each of the 48 PHYs. The entries are addressed by specifying the PHY ID of the entry that will be accessed. The bits in the Indirect Address Register, as described in Table 61, control the indirect access cycle.

This section describes the algorithms for reading and writing a PHY Mapping Table entry using the indirect access registers. The algorithms apply to all both PHY Mapping Tables.

Table 60 - Registers for PHY Map Indirect Accessing

TxL Registers	RxL Registers	Description
TxL PHY Indirect Address (0x288)	RxL PHY Indirect Address (0x208)	Address and control bits. See Table 52.
TxL PHY Indirect Data (0x289)	RxL PHY Indirect Data (0x209)	Value of the mapped PHY ID that will be written or has been read.

Table 61 - PHY Indirect Address Register (0x209, 0x289)

Bit	Name	Description
31:16	Unused	Always set to logic 0 when writing. Read value is undefined.
15	BUSY	0 : Indicates PHY Mapping Table is available for an access and value in the Data register is stable. 1 : Indicates a PHY Mapping Table Access request is pending. Do not initiate another request or read the Data register.
14	CONFIG_RWB	0 : Write request initiated when this is written. 1 : Read request initiated when this is written.
13:6	Unused	Always set to logic 0 when writing. Read value is undefined.
5:0	PHY_ADDR[5:0]	PHY ID that addresses the entry in the PHY Mapping Table that will be accessed.

7.6.1 Reading PHY ID Mapping Table Entries

1. Poll the BUSY bit in the PHY Indirect Address Register until its value is 0. This indicates that the PHY Mapping Table is ready to perform an access.
2. Write to the PHY Indirect Address Register with the following settings:
    - CONFIG_RWB bit set to logic 1
    - PHY_ADDR[5:0] set to the desired physical connection ID

    This will initiate the PHY Mapping Table read cycle and assert the BUSY bit.
3. Poll the BUSY bit in the PHY Indirect Address Register until its value is 0. This indicates that the PHY Mapping Table RAM has completed the read cycle and the PHY Mapping Table entry for is copied into the.
4. Read the PHY Indirect Data Register.

7.6.2 Writing PHY ID Mapping Table Entries

The procedure to write the PHY Mapping Table entry for a given physical connection is as follows. This algorithm applies to both the TxL and RxL Mapping Table by using either the TxL registers or the RxL registers.

1. Poll the BUSY bit in the PHY Indirect Address Register until its value is 0. This indicates that the PHY Mapping Table is ready to perform an access.
2. Write the desired PHY Mapping Table entry value to the PHY Indirect Data Register.
3. Write to the PHY Indirect Address Register with the following settings:
    - CONFIG_RWB bit set to logic 0
    - PHY_ADDR[5:0] set to the desired physical connection ID

    This will initiate the PHY Mapping Table read cycle and assert the BUSY bit. When the BUSY bit is deasserted the PHY Mapping Table write is complete.

7.7 PHY Polcing RAM Interfacing

Access to the Per-PHY Policing Configuration Tables in the Per-PHY Policing RAM is provided indirectly through the registers listed in Table 62. The Per-PHY Policing RAM has 48 PHY Policing Configuration Tables that correspond to each of the 48 PHYs. The entries are addressed by specifying the PHY ID of the configuration table that will be accessed. The bits in the PHY Policing RAM Address and Access Control Register, as described in Table 63, control the indirect access cycle.

This section describes the algorithms for reading and writing a Per-PHY Policing Configuration Table using the indirect registers.

Table 62 - Registers for PHY Policing RAM Indirect Access

Register	Address	Description
PHY Policing RAM Address and Access Control	0x144	Address and control bits.
PHY Policing RAM Data Row 0	0x145	Read or write data for per-PHY Policing RAM row 0.
PHY Policing RAM Data Row 1	0x146	Read or write data for per-PHY Policing RAM row 1.
PHY Policing RAM Data Row 2	0x147	Read or write data for per-PHY Policing RAM row 2.
PHY Policing RAM Data Row 3	0x148	Read or write data for per-PHY Policing RAM row 3.

Table 63 - PHY Policing RAM Access Control Reg (0x144)

Bit	Name	Description
31:19	Unused	Always set to logic 0 when writing. Read value is undefined.
18	RWB	0 : Write request initiated when this is written. 1 : Read request initiated when this is written.

Bit	Name	Description
17	BUSY	0 : Indicates PHY Policing RAM is available for an access and values in the Data registers are stable. 1 : Indicates a PHY Policing RAMAccess request is pending. Do not initiate another request or read the Data registers.
16	CLRONRD	0 : PHY Policing RAM entry is unchanged after a read. 1 : After a read access, the Alternate Cell Count fields of the VC Table are written to all 0's and the VC Table's CRC is updated.
15	WR_PHYCONFIG	Write masks for selected fields in the PHY Policing RAM. Set bit to logic 1 to prevent the associated field(s) from being changed during a write access.
14	WR_PHYNONCOMP3	
13	WR_PHYNONCOMP2	
12	WR_PHYNONCOMP1	
11	WR_RESERVED	
10	WR_PHYI	
9	WR_PHYL	
8	WR_PHYTAT	
7:6	Unused	Always set to logic 0 when writing. Read value is undefined.
5:0	PHY_ADDR[5:0]	PHY ID that addresses the entry in the PHY Mapping Table that will be accessed.

7.7.1 Reading PHY Policing Configuration Tables

The procedure to read the PHY Policing Configuration Table for a given physical connection is as follows:

1. Poll the BUSY bit in the PHY Policing RAM Address and Access Control Register until its value is 0. This indicates that the PHY Policing RAM is ready to perform an access.

2. Write to the PHY Policing RAM Address and Access Control Register with the following settings:

    - RWB bit set to logic 1
    - PHYAddr[5:0] set to the desired physical connection ID

    This will initiate the PHY Policing RAM read cycle and assert the BUSY bit.

3. Poll the BUSY bit in the PHY Policing RAM Address and Access Control Register until its value is 0. This indicates that the PHY Policing RAM is has completed the read cycle and the data from the Internal Per-PHY Policing RAM is copied into the PHY Policing RAM Data Row registers 0, 1, 2 and 3.

4. Read the PHY Policing RAM Data Row registers 0, 1, 2 and 3 as desired.

7.7.2 Writing PHY Policing Configuration Tables

The procedure to write the PHY Policing Configuration Tables for a given physical connection is as follows:

1. Poll the BUSY bit in the PHY Policing RAM Address and Access Control Register until its value is 0. This indicates that the PHY Policing RAM is ready to perform an access.

2. Write the appropriate values to PHY Policing RAM Data Row registers 0, 1, 2 and 3.

    NOTE: If this is the initial setup then ensure that the following fields of the Internal PHY Policing RAM table are set to zero: PhyTAT LSB, Reserved, PhyTAT MSB, Phy Non-Compliant2, Phy Non-Compliant1, Phy Non-Compliant3.

3. Write to the PHY Policing RAM Address and Access Control Register with the following settings:

    - RWB bit set to logic 0
    - PHYAddr[5:0] set to the desired physical connection ID
    - Write mask fields set to logic 1 for fields that are desired to be written to.

    This will initiate the PHY Policing RAM write cycle and assert the BUSY bit. When the BUSY bit is deasserted the PHY Policing RAM access is complete.

NOTE: If this is the initial setup then ensure that the masks for the fields mentioned in Step 2 are set to one. If this is not the initial setup then ensure that the write mask bits Wr_PhyTAT and Wr_Reserved are set to zero to prevent writing of the PhyTAT MSB, PhyTAT LSB, and Reserved fields.

7.7.3 Initializing Per-PHY Policing

Initialize the global settings that are shared between all the physical connection policing operations as follows:

1. Disable the per-PHY policing for all physical connections by writing zeroes to the PHY Policing Enable 1 Register and the PHY Policing Enable 2 Register.

2. Write the four GCRA configurations to the PHY Policing Configuration Register. Each PHY Policing instance will choose from one of these.

3. Write the three non-compliant cell count configurations to the Per-PHY Non-Compliant Cell Counting Configuration Register.

7.8 PHY Count Interfacing

Access to the cell count entries in the PHY Count RAM is provided indirectly through the registers listed in Table 64. The PHY Count RAM is read-only and has 48 sets of count entries that correspond to each of the 48 PHYs. The entries are addressed by specifying the PHY ID of the entry that will be accessed. The bits in the Per-PHY Control Register, as described in Table 65, control the indirect access cycle.

This section describes the algorithms for reading the cell counts for a given PHY using the indirect access registers.

Table 64 - Registers for PHY Policing RAM Indirect Access

Register	Address	Description
Per-PHY Counter Control	0x1A1	Address and control bits. See Table 65.
Per-PHY CLP0 Cell Count Holding Register	0X1A8	Data registers that are loaded with the per-PHY counts after a read is initiated by the Per-PHY Counter Control Register.
Per-PHY CLP1 Cell Count Holding Register	0X1A9	
Per-PHY Valid RM Cell Counts Holding Register	0X1AA	
Per-PHY Valid OAM Cell Counts Holding Register	0X1AB	
Per-PHY Errored OAM/RM Cell Counts Holding Register	0X1AC	
Per-PHY Invalid VPI/VCI/PTI Cell Counts Holding Register	0X1AD	
Per-PHY EFCI/Non-Zero GFC Cell Count Holding Register	0X1AE	
Per-PHY Timed-Out Cell Count Holding Register	0X1AF	
Per-PHY Lsat Unknown VPI & VCI Holding Register	0X1B0	

Table 65 - Per-PHY Counter Control Register (0x1A1)

Bit	Name	Description
31:18	Reserved	Always set to logic 0 when writing. Read value is undefined.
17	RWB	0 : Write request initiated when this is written. 1 : Read request initiated when this is written.
16	BUSY	0 : Indicates the PHY Counts are available for an access and values in the Holding registers are stable. 1 : Indicates a PHY Counts request is pending. Do not initiate another request or read the Holding registers.
15	CLP0_CLRONRD	Clear On Read bits for each of the PHY Count data registers. selected fields in the PHY Policing RAM. Each bit can be set individually as follows: Set to logic 0 to cause the associated Count Data register to be remain unchanged after a read accessed is performed. Set to logic 1 to cause the associated Count Data register to be written to all ‘0’s after a read accessed is performed.
14	CLP1_CLRONRD	
13	RM_CLRONRD	
12	OAM_CLRONRD	
11	INVOAMRM_CLRONRD	
10	INVAL_CLRONRD	
9	NZGFC_CLRONRD	
8	TO_CLRONRD	
7:6	Reserved	Always set to logic 0 when writing. Read value is undefined.
5:0	PHYID[5:0]	Specifies the PHY for which the counts will be accessed.

7.8.1 Reading Per-PHY Cell Counts

The procedure to read the Per-PHY Cell Counts for a given physical connection is as follows:

1. Poll the BUSY bit in the Per-PHY Counter Control Register until its value is 0. This indicates that the Per-PHY Counters are ready to perform a read.

2. Write to the Per-PHY Counter Control Register with the following settings:

    - RWB bit set to logic 1
    - Clear on read bits set to logic 1 for all fields that are desired to be cleared.
    - PHY_ADDR[5:0] set to the desired physical connection ID

    This will initiate the Per-PHY Counter read cycle and assert the BUSY bit.

3. Poll the BUSY bit in the Per-PHY Counter Control Register until its value is 0. This indicates that the read cycle has completed and all the Cell Counts Holding Registers contain the valid count values.

4. Read the Cell Counts Holding registers as desired.

7.8.2 Writing Per-PHY Cell Counts

The procedure to write the Per-PHY Cell Counts for a given physical connection is as follows:

1. Poll the BUSY bit in the Per-PHY Counter Control Register until its value is 0. This indicates that the PHY Mapping Table is ready to perform an access.

2. Write the desired cell count values into the Cell Count Holding registers. Note that all of the registers will be written.

3. Write to the Per-PHY Counter Control Register with the following settings:

    - RWB bit set to logic 0
    - PHY_ADDR[5:0] set to the desired physical connection ID

    This will initiate the Per-PHY Counter read cycle and assert the BUSY bit. When the BUSY bit is deasserted the Per-PHY Cell Counts register write is complete.

7.9 Calendar Entry Interfacing

Access to the entries in the three Calendar data structures is provided indirectly through the registers listed in Table 66. Each Calendar Table has 128 entries that are addressed by specifying the index number. The bits in the Calendar Indirect Address and Data Register, as described in Table 67, control the indirect access cycle and contain the data field.

This section describes the algorithms for reading and writing a Calendar entry using the indirect registers. The algorithms apply to all three Calendars.

Table 66 - Registers for Calendar Entry Access

Register	Address	Description
RxL Calendar Indirect Address and Data	(0x20B)	Indirect address field, data field, busy bit and read write bit. Calendar entries programmed indirectly through this register.
RxP Calendar Indirect Address and Data	(0x265)	
TxL Calendar Indirect Address and Data	(0x28B)	

Table 67 - Calendar Addr. and Data Reg.(0x20B, 0x265, 0x28B)

Bit	Name	Description
31:16	Unused	Always set to logic 0 when writing. Read value is undefined.
15	BUSY	0 : Indicates PHY Policing RAM is available for an access and values in the Data registers are stable. 1 : Indicates a PHY Policing RAMAccess request is pending. Do not initiate another request or read the Data registers.
14:8	CALENDAR_ADDR[6:0]	Index that addresses the entry in the Calendar that will be accessed.

Bit	Name	Description
7	CONFIG_RWB	0 : Write request initiated when this is written. 1 : Read request initiated when this is written.
6	Unused	Always set to logic 0 when writing. Read value is undefined.
5:0	CALENDAR_DATA[5:0]	Data that will be written to or has been read.

7.9.1 Reading Calendar Entry

The Calendar entries are accessed by specifying the index of the calendar table in the CALENDAR_ADDR[6:0] field of the Indirect Address and Data.

The general procedure to read a Calendar entry is as follows:

1. Poll the BUSY bit in the Calendar Indirect Address and Data Register until its value is 0. This indicates that the Calendar is ready to perform an access.

2. Write to the Calendar Indirect Address and Data Register with the following settings:

    - CONFIG_RWB set to logic 1
    - CALENDAR_ADDR[6:0] set to the table index to be read

    This will initiate the Calendar read cycle and assert the BUSY bit.

3. Poll the BUSY bit in the Calendar Indirect Address and Data Register until its value is 0. This indicates that the SDQ is has completed the read cycle and the value of CALENDAR_DATA[5:0] is valid.

7.9.2 Writing Calendar Entry

The general procedure to write a Calendar entry is as follows:

1. Poll the BUSY bit in the Calendar Indirect Address and Data Register until its value is 0. This indicates that the Calendar is ready to perform an access.

2. Write to the Calendar Indirect Address and Data Register with the following settings:

- CONFIG_RWB set to logic 0
- CALENDAR_ADDR[6:0] set to the table index to be written
- CALENDAR_DATA[5:0] set the PHY ID that to write to that entry

This will initiate the Calendar read cycle and assert the BUSY bit. When the BUSY bit is deasserted the Calendar Entry access is complete.

7.10 MCIF Interfacing

Access to ATM cells in the two Microprocessor Cell Interfaces (MCIF) is provided indirectly through the registers listed in Table 68. The Input Microprocessor Cell Interface (IMCIF) contains a one ATM cell buffer and the Output Microprocessor Cell Interface has a FIFO that holds 16 ATM cells. The cells are stored as 64 byte entries that are accessed sequentially through the Microprocessor Cell Data Register. One access operation transfers 4 bytes, so to transfer an entire cell 16 read or write operations are required. Data transfers are completed during one clock cycle, therefore there is no BUSY signal for the MCIF interface. The bits in the Microprocessor Cell Interface Control and Status Register described in Table 69 control the insertion of cells into the IMCIF, and the bits described in Table 70 control the extraction of cells from the OMCIF.

This section describes the algorithms for reading and writing an ATM cell using the indirect access registers.

Table 68 - Registers for MCIF Interfacing

Register	Address	Description
Microprocessor Cell Interface Control and Status Register	(0x020)	Control bits.
Microprocessor Cell Data Register	(0x021)	Indirect data register.

Table 69 - MCIF Control and Status Reg(0x020) Insert Bits

Bit	Name	Description
31:27	Unused	Always set to logic 0 when writing. Read value is undefined.
26	INSRDY	0 : Indicates PHY Policing RAM is available for an access and values in the Data registers are stable. 1 : Indicates a PHY Policing RAMAccess request is pending. Do not initiate another request or read the Data registers.
25	WRSOC	0 : Write request initiated when this is written. 1 : Read request initiated when this is written.

Bit	Name	Description
24:19	PHY_ID[5:0]	Specifies the PHY ID that will be associated with the cell that is inserted. If PROC_CELL is set to logic 1 this PHY ID will used to construct the routing word for the search tree lookup in the cell processor. If PROC_CELL is set to logic 0 the cell will be inserted directly into the output stream of this PHY.
18	PROC_CELL	0 : The cell written to the IMCIF will be sent cell directly to the Output Cell Interface without being processed. 1 : The cell written to the IMCIF will be inserted into the cell stream as if it had come from the Input Cell Interface. The cell will be handled by the cell processor in the same manner as all other cells coming from the ICIF.
17	CRC10	0 : 1 : The last 10 bits of the inserted cell will be replaced by a CRC code. This bit must be set during all the 16 indirect writes used to assemble the cell. Typically the last 2 bytes (16 bits) of the cell should be written to zero, and the CRC will replace the 10 least significant bits.
0	INSRST	0 : IMCIF operates normally 1 : IMCIF in reset state. The cell insertion FIFO is cleared, and writes to the FIFO will have no effect.

Table 70 - MCIF Control and Status Reg(0x020) Extract Bits

Bit	Name	Description
15:6	Unused	Always set to logic 0 when writing. Read value is undefined.
5	RSOC	0 : Indicates the Microprocessor Cell Data Register does not contain the first word of a cell. 1 : Indicates that the Microprocessor Cell Data Register currently contains the first word of a cell. Check this bit at the beginning of a read cycle to ensure that the first word is being read.
4	EXTCA	0 : Indicates that there are no cells in the OMCIF. The transition from 1 to 0 is caused when the last word of the last cell in the OMCIF is read. 1 : Indicates that there is at least one complete cell in the OMCIF. The transition from 0 to 1 is caused when the OMCIF is empty and receives the last byte of the first cell. This bit sets the UPCAI bit in the Master Interrupt Status #1 Register (0x002) which can be used for interrupt driven waiting for cell arrivals.
3	ABORT	Set to logic 1 to discard the current cell in the OMCIF. If there is another cell in the OMCIF, the next cell will become available and the cell read pointer will be reset to the first word.
2	RESTART	Set to logic 1 to reset the cell read pointer in the OMCIF to the first word of the current cell. Cells are not discarded from the OMCIF FIFO until the last word has been read, so when this is set the current cell will not be affected and the 16 word read sequence can begin again.
1	DMAREQINV	0 : External signal UP_DMAREQ is active high. 1 : External signal UP_DMAREQ is active low.
0	EXRST	0 : OMCIF operates normally 1 : OMCIF in reset state. The cell extraction FIFO is cleared, and no cells are accepted.

7.10.1 Reading Cells

The operation to read a cell will typically be called when the software has detected that a cell is available using one of the methods described in section 6.8.1. To read the next available cell from the OMCIF FIFO, use the following procedure:

1. Check that the EXTCA bit is logic 1. Do not proceed if the EXTCA bit logic 0.

2. Write to the Microprocessor Cell Interface Control and Status Register with the following settings:

    - RESTART = 0

    This initiates a read from the OMCIF that will continue for 16 successive read operations from the Microprocessor Cell Data Register

3. Read the Microprocessor Cell Interface Control and Status Register and confirm that RSOC is logic 1. This indicates that the data word in the Microprocessor Cell Data Register is the first word of the cell.

4. Read the Microprocessor Cell Data Register 16 times to read all the words of the cell. The first word read is the first word of the cell and each subsequent word increments sequentially through the cell as specified in the Data Sheet [1].

7.10.2 Writing Cells

To write a cell to the IMCIF FIFO, use the following procedure:

1. Check that the INSRDY bit is logic 1. Do not proceed if INSRDY is logic 0.

2. Write to the Microprocessor Cell Interface Control and Status Register with the following settings:

    - RESTART = 0
    - WRSOC = 1
    - CRC10, PROC_CELL, PHYID[5..0] = application specific values

    Setting RESTART initiates a write to the IMCIF that will continue for 16 successive write operations to the Microprocessor Cell Data Register

3. Write to the Microprocessor Cell Data Register 16 times to write all the words of the cell. The data words are written in the sequence specified in the Data Sheet [1].

7.11 Change of State FIFO Interfacing

Access to the entries in the Change of State (COS) FIFO is provided indirectly through the registers listed in Table 71. The COS FIFO is read-only and has 256 entries that are accessed sequentially. Entries are written to the COS FIFO by a background process within the ATLAS-3200. The bits in the VC Table Change of Connection State FIFO Status Register, as described in Table 72, provide the status bits used for reading from the FIFO.

This section describes the algorithms for reading the first entry from the COS FIFO using the indirect access registers.

Table 71 - Registers for COS FIFO Interfacing

Register	Address	Description
VC Table Change of Connection State FIFO Status	0x190	Status bits
VC Table Change of Connection State FIFO Data	0x191	COS FIFO data register. Automatically updated to contain the next COS FIFO entry after each read.

Table 72 - VC Table COS FIFO Status Register (0x190)

Bit	Name	Description
31:3	Unused	Always set to logic 0 when writing. Read value is undefined.
2	COSFULL	0 : Indicates that the COS FIFO is not full. 1 : Indicates that the COS FIFO is full. No more entries will be accepted and changes of state may be missed. The microprocessor can use the COSI, XCOSI, and COSFULLI interrupt bits in the Master Interrupt Status #1 Register to respond to cell levels in the COS FIFO and prevent it from getting full.

Bit	Name	Description
1	COSVALID	Check this bit before reading from the VC Table Change of Connection State FIFO Data Register. 0 : The VC Table Change of Connection State FIFO Data Register does not contain valid data. This may be due to the COS FIFO being empty or due to a read access of the next entry still in progress. 1 : There is a valid entry COS FIFO in the VC Table Change of Connection State FIFO Data Register. and it may be read.
0	COSBUSY	Poll this bit to determine when the COSVALID is valid and can checked 0 : The COSVALID bit is valid. 1 : Indicates that the COS FIFO read pointer is being updated and the value of COSVALID is not valid.

7.11.1 Reading COS Entries

The operation to read a COS FIFO entry will typically be called when the software has detected that the FIFO is not empty by servicing one of the COS FIFO interrupts described in Section 6.9.1.

To read the next entry from the COS FIFO, use the following procedure:

1. Poll the COSBUSY bit in the VC Table Change of Connection State FIFO Status Register until its value is logic 0. This indicates that the COS FIFO has updated its read pointer.

2. Poll the COSVALID bit in the VC Table Change of Connection State FIFO Status Register until its value is 0. This indicates that the VC Table Change of Connection State FIFO Data register contains a valid COS FIFO entry.

3. Read the VC Table Change of Connection State FIFO Data Register to get the COS FIFO entry. If the COS FIFO contains subsequent entries then the COSBUSY bit will be asserted and the COS FIFO will shift its entries and load the new top entry into the VC Table Change of Connection State FIFO Data Register.

7.12 Count Rollover FIFO Interfacing

Access to the entries in the Count Rollover (CRO) FIFO is provided indirectly through the registers listed in Table 73. The CRO FIFO is read-only and has 256 entries that are accessed sequentially. Entries are only written to the CRO FIFO by a background process within the ATLAS-3200. The bits in the VC Table Change of Connection State FIFO Status Register, as described in Table 74, provide the status bits used for reading from the FIFO.

This section describes the algorithms for reading the first entry from the CRO FIFO using the indirect access registers.

Table 73 - Registers for CRO FIFO Interfacing

Register	Address	Description
VC Table Count Rollover FIFO Status	0x198	Status bits
VC Table Count Rollover FIFO Data	0x199	Indirect data register

Table 74 - VC Table CRO FIFO Status Register (0x198)

Bit	Name	Description
31:3	Unused	Always set to logic 0 when writing. Read value is undefined.
2	CRFULL	0 : Indicates that the CRO FIFO is not full. 1 : Indicates that the CRO FIFO is full. No more entries will be accepted and changes of state may be missed. The microprocessor can use the CROI, XCROI, and CROFULLI interrupt bits in the Master Interrupt Status #1 Register to respond to cell levels in the CRO FIFO and prevent it from getting full.

Bit	Name	Description
1	CRVALID	Check this bit before reading from the VC Table Change of Connection State FIFO Data Register. 0 : The VC Table Change of Connection State FIFO Data Register does not contain valid data. This may be due to the CRO FIFO being empty or due to a read access of the next entry still in progress. 1 : There is a valid entry CRO FIFO in the VC Table Change of Connection State FIFO Data Register. and it may be read.
0	CRBUSY	Poll this bit to determine when the CROVALID is valid and can checked 0 : The CROVALID bit is valid. 1 : Indicates that the CRO FIFO read pointer is being updated and the value of CROVALID is not valid.

7.12.1 Reading CRO Entries

The operation to read a CRO FIFO entry will typically be called when the software has detected that the FIFO is not empty by servicing one of the CRO FIFO interrupts described in Section 6.9.2. To read the next entry from the CRO FIFO, use the following procedure:

1. Poll the CROBUSY bit in the VC Table Change of Connection State FIFO Status Register until its value is 0. This indicates that the CRO FIFO has updated its read pointer.

2. Poll the CROVALID bit in the VC Table Change of Connection State FIFO Status Register until its value is 0. This indicates that the VC Table Change of Connection State FIFO Data register contains a valid CRO FIFO entry.

3. Read the VC Table Change of Connection State FIFO Data Register to get the CRO FIFO entry. If the CRO FIFO contains subsequent entries then the CROBUSY bit will be asserted and the CRO FIFO will shift its entries and load the new top entry into the VC Table Change of Connection State FIFO Data Register.

8 PSEUDO-CODE REFERENCE

To present the concepts as clearly as possible, the pseudo-code routines are simplified to demonstrate only the basic functional requirements. The major modifications required to adapt the pseudo-code to an actual code implementation are listed below:

- Error Handling: There is limited error checking and error handling in the pseudo-code and many of the routines do not have return values to indicate a success or failure. The routines should be modified to check for error conditions and return success/failure values

- Multiple Device Support: All of the pseudo-code routines are intended to perform operations only on a single ATLAS-3200. Since typical applications will have more than one, it may be desired to modify the routines so they can be used for multiple devices.

  For example, the regRead routine reads a register value from an ATLAS-3200 that is assumed to be memory mapped to a constant base address.

```
UINT4 regRead(
    UINT2 regAddr)
{
UINT4 address
UINT4 value

address = ATLAS_BASE_ADDR + regAddr
value = *(address)
return value
}
```

  To modify this routine to accommodate multiple ATLAS-3200s memory mapped to various locations, an additional input parameter specifying which ATLAS-3200 device to read from can be added.

```
DEVICE_BASE_ADDR[] = {ATLAS1_BASE_ADDR, ATLAS2_BASE_ADDR}

UINT4 regRead(
    UINT2 regAddr
    UINT1 device)
{
UINT4 address
UINT4 value

address = DEVICE_BASE_ADDR[device] + regAddr
```

```
value = *(address)
return value
}
```

- Global Variables: Global variables are used by the pseudo-code to store the mirror data structures. The names of global data structures are prefixed with ‘g’. The use of these data structures can be modified so that instead of accessing them globally, pointers to the data structures are passed to each function that requires them. This is required for functions to support multiple devices since the microprocessor must maintain separate mirror copies for each ATLAS-3200.

  e.g.

```
/* Shadow copy of the Primary Search Table in the ATLAS-3200 */
/* external SRAM. */
UINT4 gPrimarySearchTable[]
```

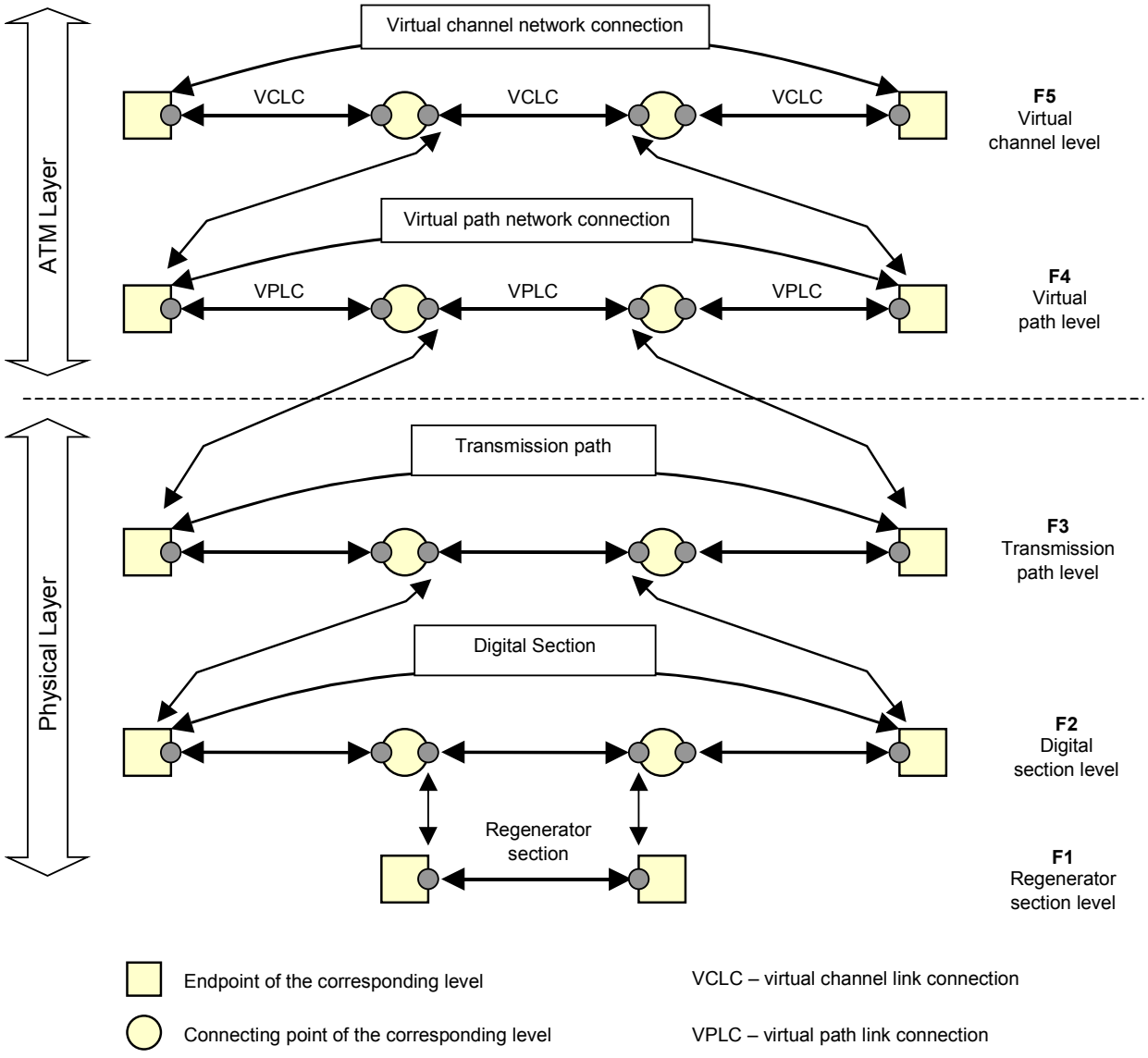
9 APPENDIX A: OAM CELL DESCRIPTIONS

This appendix contains a general description of the ATM Operations and Maintenance (OAM) cell flows. It provides background information for the OAM Cell Management section and serves as a general reference for understanding the OAM flow.

The OAM functions in the network are performed on five OAM hierarchical levels associated with the ATM and physical layers of the protocol reference model. The functions result in corresponding bidirectional information flows on five different levels. A pictorial explanation of the levels is shown in Figure 47 below. The levels are as follows:

F5, Virtual channel (VCC) level	Extends between network elements performing virtual channel connection termination and is shown extending through one or more virtual paths.
F4, Virtual path (VPC) level	Extends between network elements performing virtual path connection termination and is shown extending through one or more transmission path.
F3, Transmission path level	Extends between network elements assembling/disassembling the payload of a transmission system and associating it with its OAM functions.
F2, Digital section level	Extends between section endpoints and comprises a maintenance entity.
F1, Regenerator section level	A regeneration section (e.g. usually SONET optical interface) is a portion of a digital section and as such is a maintenance sub-entity.

Figure 47 - ATM OAM Hierarchical Levels



NOTE: the F5 flow does not necessarily extend beyond the end point of F4 flow.

Levels F1, F2 and F3 are physical layer (SONET) OAM flows, while F4 and F5 flows are the ATM layer OAM flows. ATM Layer OAM flows are necessary, since a virtual connection may extend beyond the SONET network. The ATLAS-3200 only handles the F4 and F5 ATM Layer OAM flows.

F4 flows refer to OAM flows over a VPC. F5 flows refer to OAM flows over a VCC. An OAM flow may extend over the entire connection. This is called an

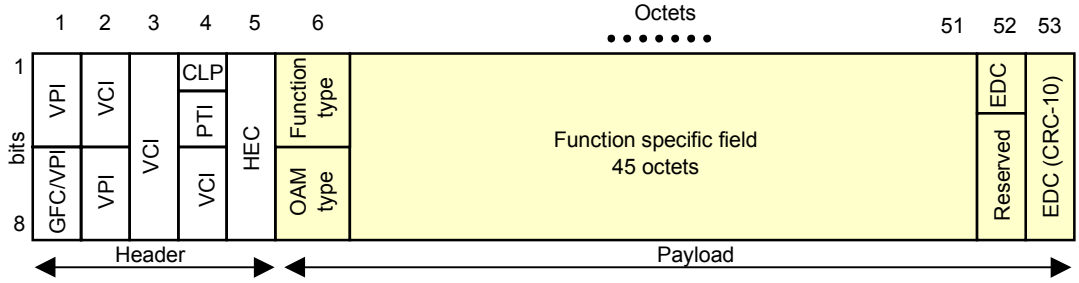
end-to-end flow. An OAM flow may also cover a portion of the end to end connection. This is called a segment flow. Therefore, there are four main types of OAM flows:

- F4 End-to-end
- F4 Segment
- F5 End-to-end
- F5 Segment

9.1 General OAM Cell Format

Figure 48 below shows an ATM OAM Cell. The total size of any ATM Cell (including OAM cell) is 53 octets. In the OAM cell, the forty-eight octets of the user payload contain OAM specific data.

Figure 48 - OAM Cell Structure



The OAM cell is distinguished from a user cell by the VCI or PTI header fields. F4 OAM cells are identified by the VCI field as highlighted in Table 75. F5 OAM cells are identified by the PTI field as highlighted in Table 76.

Table 75 - F4 OAM Cell Identification

VPI	VCI	Interpretation	Category
same as user cells	0	Unassigned cell (VPI=0)	Non-User
	0	Unused (VPI>0)	
	1	Meta-signaling cell (UNI)	User
	2	General broadcast signaling cell (UNI)	
	3	Segment OAM F4 flow call	Non-User
	4	End-to-end OAM F4 flow call	
	5	Point-to-point signaling cell	User
	6	Resource management cell	Non-User
	7-15	Reserved for future use.	
	16-31	Reserved for future use.	User
	>31	Available for user data transmission	

Table 76 - F5 OAM Cell Identification

VPI	VCI	PTI	Interpretation	Category
same as user cells	same as user cells	000	User data cell, congestion not experienced	User
		001		
		010	User data cell, congestion experienced	
		011		
		100	Segment OAM F5 flow call	Non-User
		101	End-to-end OAM F5 flow call	
		110	Resource management cell	Non-User
		111	Reserved for future use	

The cell payload of an OAM cell contains the OAM specific data. The first byte of the cell payload consists of two fields that indicate the OAM cell type and OAM function type. There are 4 cell types defined: Fault Management (FM), Performance Management (PM), Activation/Deactivation (A/D), and System Management (SM). For each cell type, there are several Function Types. The OAM cell types and OAM function types are detailed in Table 77.

Table 77 - OAM Cell Types and Functions

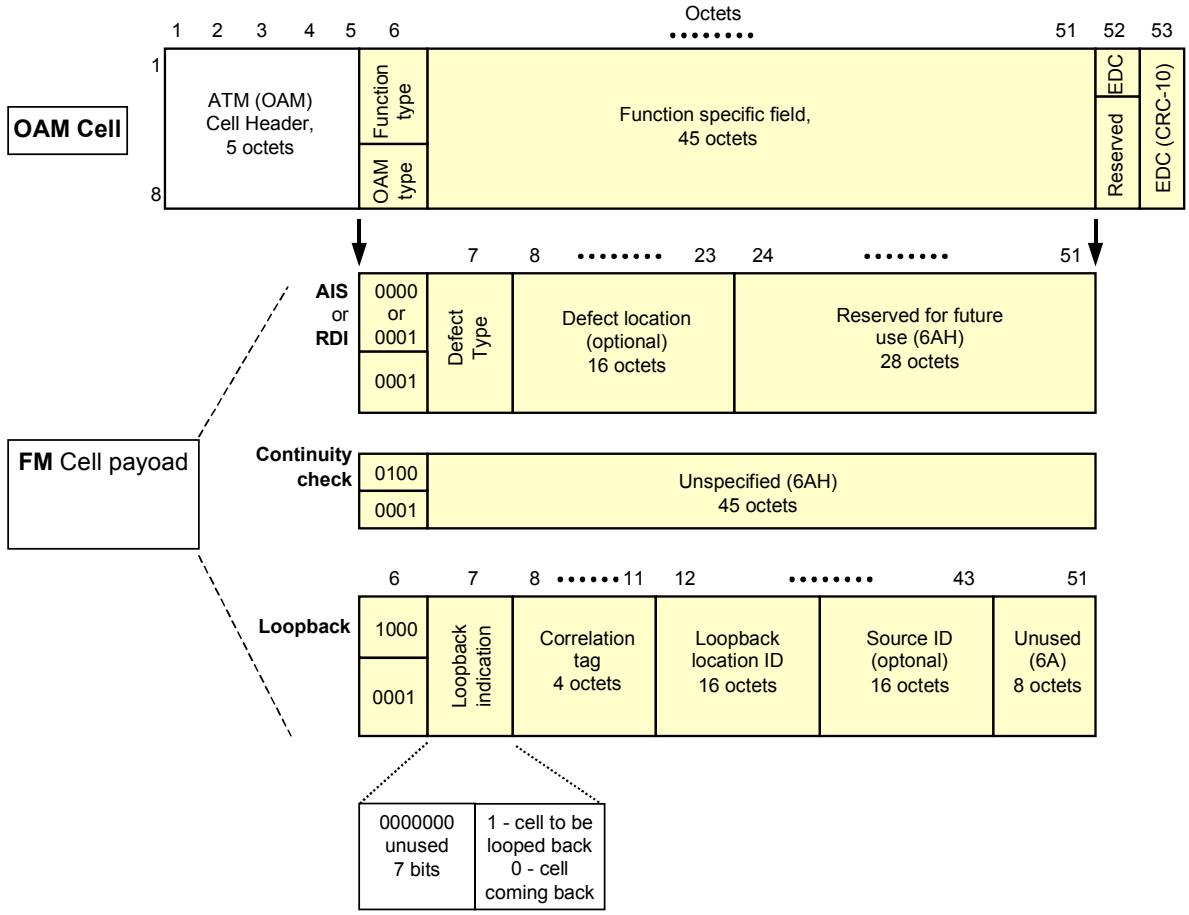
OAM Cell Type		OAM Function Type		Description
Fault Management (FM)	0001	AIS (alarm indication signal)	0000	For reporting defect indications in the forward direction
		RDI (remote defect indication)	0001	For reporting remote defect indications in the backward direction
		CC (Continuity check)	0100	For continuously monitoring the availability of a link
		LB (Loopback)	1000	For on-demand connectivity monitoring, fault localization, and pre-service connectivity verification
Performance Management (PM)	0010	Forward Performance Monitoring	0000	For estimating performance over a link or segment of a link
		Backward reporting	0001	For reporting performance estimations on the backward direction
Automated Protection Switching Coordination protocol (APS)	0101	Group Protection	0000	For carrying group protection switching protocol information
		Individual Protection	0001	For carrying individual protection switching protocol information
Activation/ Deactivation (A/D)	1000	Performance Management A/D Forward Performance Monitoring and Backward Reporting	0000	For activation/deactivation of PM functionality in a standard way
		Continuity check A/D	0001	For activation/deactivation of CC functionality in a standard way
		Forward monitoring A/D	0010	For activation/deactivation of FM functionality in a standard way
System Managemen (SM)	1111	Not specified in I.610 (1999)		For use by end systems only.

Ten bits at the end of the OAM cell are dedicated to the Error Detection Code (EDC). The EDC is calculated over the cell payload. The EDC is used protect against erroneous decisions based on corrupted OAM cell data.

9.2 Fault Management (FM) Cells

The three types of Fault Management (FM) OAM cells are shown in Figure 49.

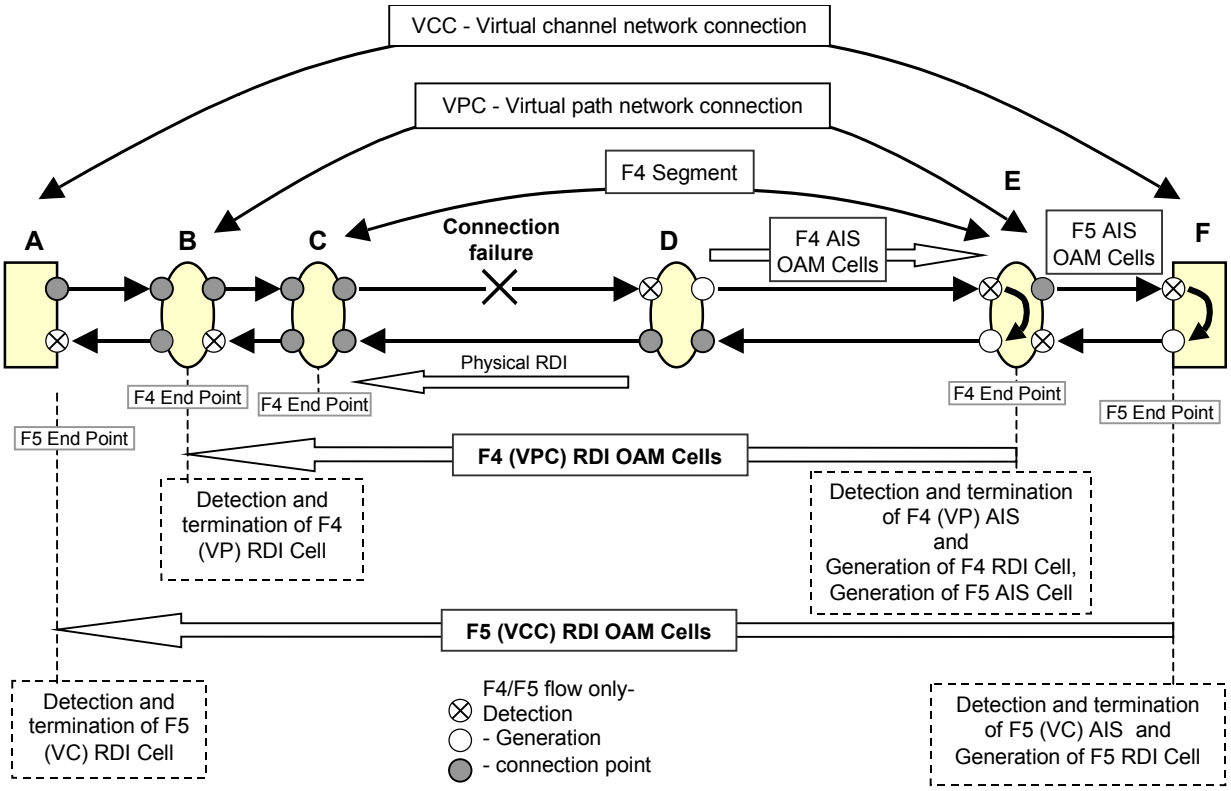
Figure 49 - FM Cell Function Specific Fields



9.2.1 AIS and RDI Cells

AIS cells are used to report defects in the downstream direction. RDI cells are used to report defects in the upstream direction. An example of the AIS and RDI flow is shown in Figure 50 below.

Figure 50 - AIS and RDI Flow



The example shown above has the OAM flows set as follows:

- "A" to "F" is an F5 End-to-End flow
- "B" to "E" is an F4 End-to-End flow
- "C" to "E" is a Segment F4 flow
- "D" is a connection point (non-end point to either flow).

Consider a failure between "C" and "D". The network will react as follows:

The failure is detected at "D". It may be detected at the physical layer or at the ATM layer.

- "D" generates Segment and End-to-End F4 AIS cells downstream once per second. "D" does not generate RDI nor F5 AIS, since it is not an end-point for the F4 or F5 flows. "D" may generate physical layer RDI, if appropriate.

- "E" is both a segment and end-to-end flow end point, and therefore terminates the F4 AIS cell flow. In response to the AIS cells, "E" generates F4 Segment and End-to-End RDI cells upstream. Additionally, since E is also a connecting point for the F5 flow, F5 End-to-End AIS cells are sent downstream once per second.

- "F" is end-to-end flow end point, and therefore terminates the F5 End-to-End AIS. In response to the AIS, "F" generates F5 End-to-End RDI cells upstream.

- "E" does not terminate the F5 End-to-End RDI generated at "F", since it is not an end point for the F5 OAM flow.

- "D" does not terminate OAM cells since it is not an end point.

- "C" terminates the F4 Segment RDI generated by "E".

- "B" terminates the F4 End-to-End RDI generated by "E".

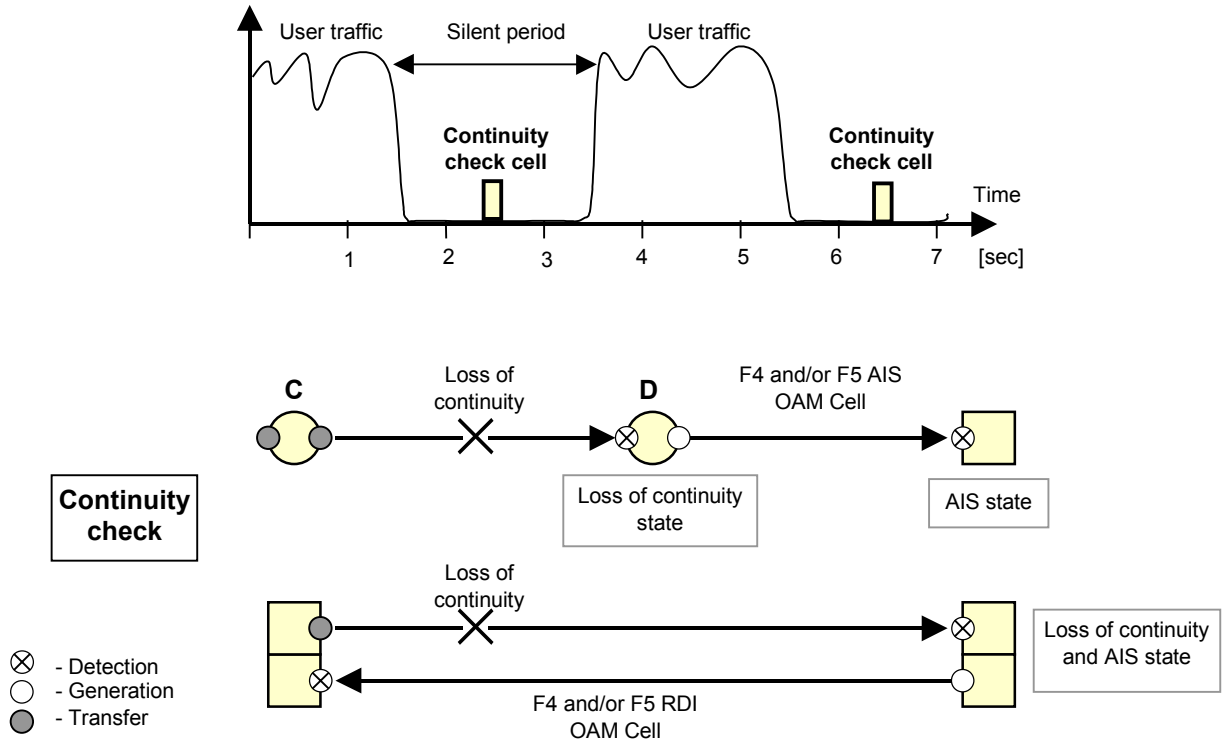
- "A" terminates the F5 End-to-End RDI cells generated by "E".

- "A", "B" and "C" do not generate any OAM cells related to this connection failure.

9.2.2 CC Cells

Continuity failures at the ATM layer are detected using Continuity Check (CC) cells. Continuity check cells are inserted into a connection so the downstream entity may differentiate between a loss of continuity and a period of low cell flow.

Figure 51 - CC Flow

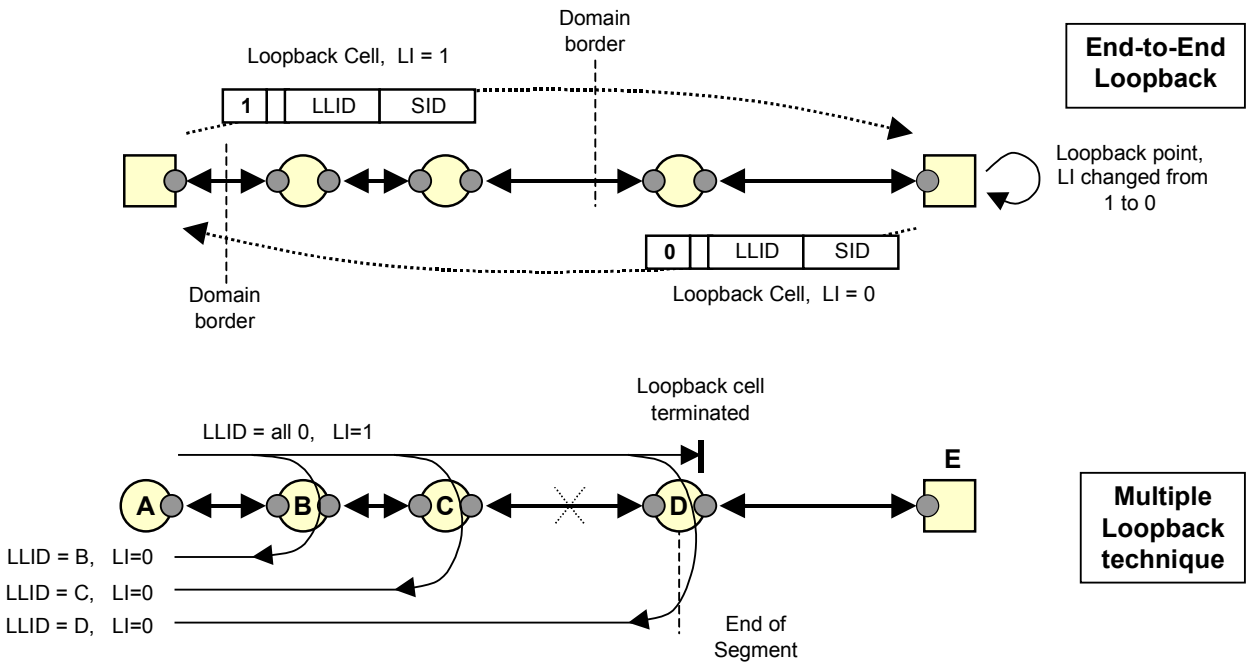


Connection points (intermediate nodes) in a segment and/or end-to-end OAM flow (F4 and F5) can be set to look for the presence of CC cells. If a lack of user cells and CC cells is detected over 3.5 ±0.5sec, a network entity may trigger CC alarm, which in turn may activate sending of AIS cells downstream. The CC alarm triggered at the flow end-points may also activate sending RDI cells in upstream direction.

9.2.3 Loopback Cells

Another type of fault management cell is the loopback cell. The intent of the loopback cell is to determine continuity in a connection after it has been setup, and to isolate misconfiguration problems. This is a demand service used by network operators. Examples of loopback flows are shown in Figure 52 below.

Figure 52 - Loopback Flow Examples



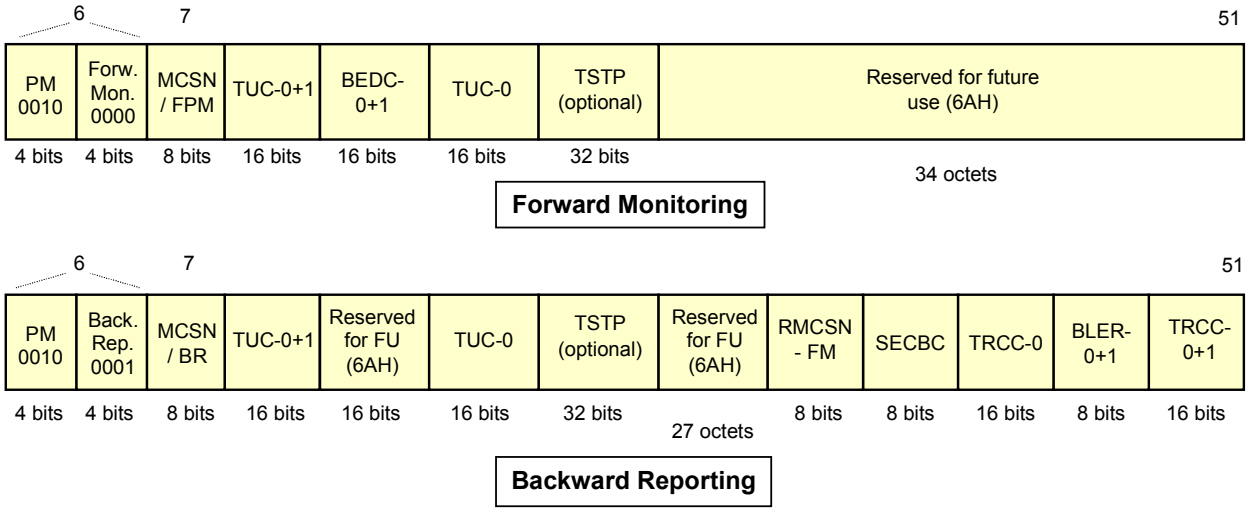
The simplest use of the loopback cell is shown in the top half of Figure 52. The loopback cell is sent from one end of the connection and looped back at the end of the connection.

A multiple loopback technique can be used to simplify the segment diagnostic process (as shown above). In this case the Loopback Location ID (LLID) field of a cell generated at point "A" is filled with all 0's. Each connection point in the segment sends the loopback cell back with the LLID changed to that node's ID and loopback indicator (LI) changed to 0. The loopback cell originating point "A" can receive, for example, three cells. If connection C↔D is broken, point A receives only two loopback cells back (B and C). The segment loopback cell is terminated at the segment end point (D).

9.3 Performance Management (PM) Cells

Performance Management (PM) cells are used to determine the performance of a particular VC. PM is generally a demand service initiated by the network operator. Two types of PM cells are defined: forward monitoring and backward reporting. The function specific fields of the PM cell are shown in Figure 53 below.

Figure 53 - PM Cell Function Specific Fields



A full description of each field is presented in I.610. A brief summary is presented below.

Forward monitoring:

MCSN/FPM	Monitoring Cell Sequence Niumber, forward PM
TUC0+1	Total User Cells, CLP0+1
BEDC0+1	Block Error Detection Code, CLP0+1
TUC0	Total User Cells, CLP0
TSTP	Timestamp

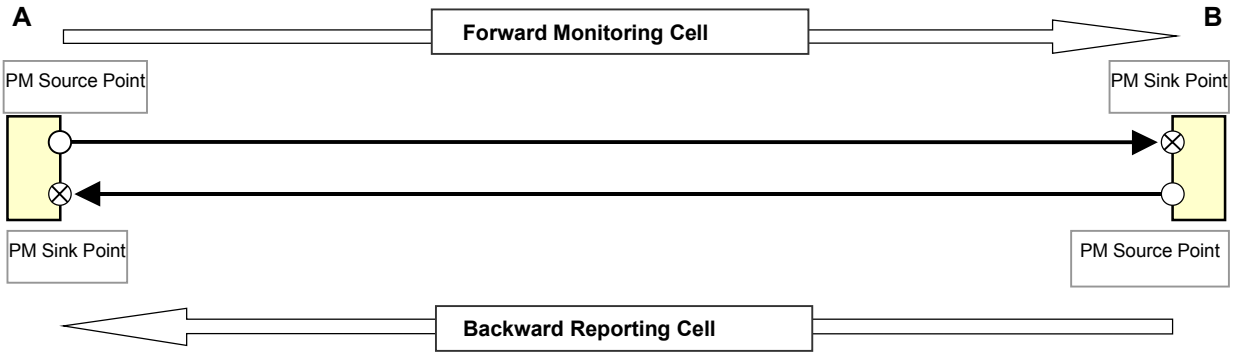
Backward reporting:

MCSN/BR	Monitoring Cell Sequence Niumber, Backward Monitoring Cell
TUC0+1	Total User Cells, CLP0+1
TUC0	Total User Cells, CLP0
TSTP	Timestamp

RMCSN-FM	Reported Monitoring Cell Sequence Number, Forward Monitoring Cell
SECBC	Severely Errored Cell Block Count
TRCC0	Total Received Cell Count, CLP0
BLER0+1	Block Error Result, CLP0+1
TRCC0+1	Total Received Cell Count, CLP0+1

The PM cells are sent at the source point, and terminated and processed at the sink points for segment and end-to-end flows on VPCs and VCCs. The PM cells may also be monitored at intermediate nodes. Unlike FM flows, there is no interaction between the F4 and F5 levels for PM flows. Figure 54 below shows how the PM flow works. The figure applies to F4 segment, F4 end-to-end, F5 segment, and F5 end-to-end flows.

Figure 54 - Example of PM Cell Flow



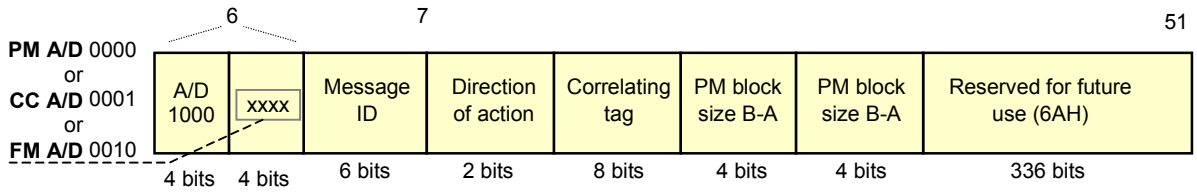
Every *n* user cells (where *n* is the block size, defined by I.610), a forward PM cell will be generated at the PM source point. The end point for the PM flow (the “sink”) will terminate the PM cell, process the contents, and generate a backward PM cell in response.

9.4 Activate / Deactivate (A/D) Cells

The Activation/Deactivation (A/D) process is use to activate and deactivate PM or CC flow sessions through the ATM network. The advantage of the A/D process is that it uses the same network that carries user traffic. At the same time, it may be a major disadvantage. For instance, if a particular connection is not performing well, it may be difficult to reliably pass A/D cells to setup a PM session.

The function specific fields of the A/D cell is shown in Figure 55 below.

Figure 55 - A/D Cell Function Specific Fields



A full description of each field is presented in I.610. A brief summary is presented below.

Message ID — Specifies the function of the cell. Refer to I.610 for the use of the message ID field.

Direction of Action — Identifies direction(s) of transmission.

Correlation Tag — A tag generated for each message, used by nodes to correlate commands with responses.

PM Block Size A-B — Specifies the A-B block size for forward monitoring.

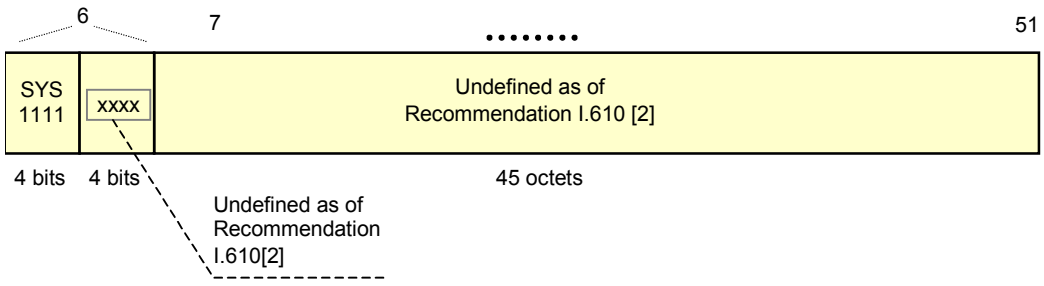
PM Block Size B-A — Specifies the B-A block size required for backward reporting. The PM block size is always 2N, where 7 □ N □ 15.

9.5 System Management (SM) Cells

The System Management cells are provided for use by end user equipment. Their use is optional and is not recommended except for specific applications which are not satisfied by other mechanisms.

The function specific fields of the System Management cell are shown in Figure 55.

Figure 56 - System Management Cell Function Specific Fields



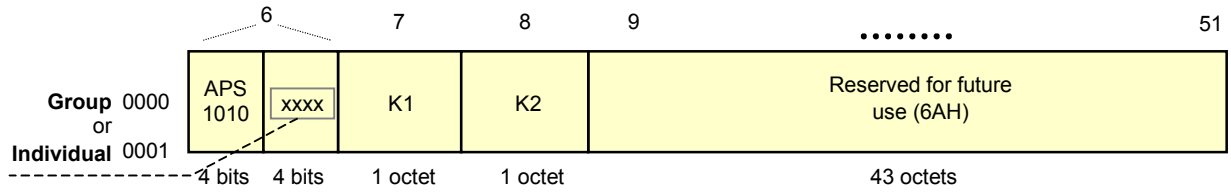
9.6 Automated Protection Switching (APS) Cells

The APS cells provide mechanisms for protection switching at the ATM layer to protect against physical layer and ATM layer defects. They apply primarily to the situations where a server layer protection switching does not exist

Two types of APS cells are defined: Individual and Group. Individual VP/VC protection switching is useful to protect only a part of VPs/VCs that need high reliability. Group protection switching facilitates fast ATM layer protection switching for a logical bundle of VP/VC network and/or subnetwork connections.

The function specific fields of the A/D cell is shown in Figure 55 below.

Figure 57 - APS Cell Function Specific Fields



A full description of APS cell usage and the K1 and K2 fields is presented in I.630.

10 APPENDIX B: VC BINARY SEARCH TREE EXAMPLE

This appendix provides a detailed example of the use of the binary search tree.

The following example illustrates how the search could be constructed and what happens when a cell enters the ATLAS-3200 cell processor. For this example, we will setup the ATLAS-3200 for eight connections. Table 78 shows the connection characteristics. We will assume that there is no prepend or postpend, so that the connection can be uniquely identified by the cell's header contents and PHY ID. For simplicity, the PHY ID and VPI for each of the eight connections is the same.

Table 78 - Example VC Characteristics

VC #	VPI	VCI	PHY ID
1	0x005	0x00B5	0
2	0x005	0x00B0	0
3	0x005	0x008F	0
4	0x005	0x0023	0
5	0x005	0x0020	0
6	0x005	0x0017	0
7	0x005	0x0016	0
8	0x005	0x0010	0

To setup the ATLAS-3200 to support these connections, the ATLAS-3200 is configured to use the VPI for the Primary Search, and the VPI/VCI for the Secondary search. The Search Engine Configuration Register (0x10B) settings to achieve this are summarized below in Table 79. These field settings correspond to a register value of 0x0031A000.

Table 79 - Example Search Engine Config. Reg. (0x10B)

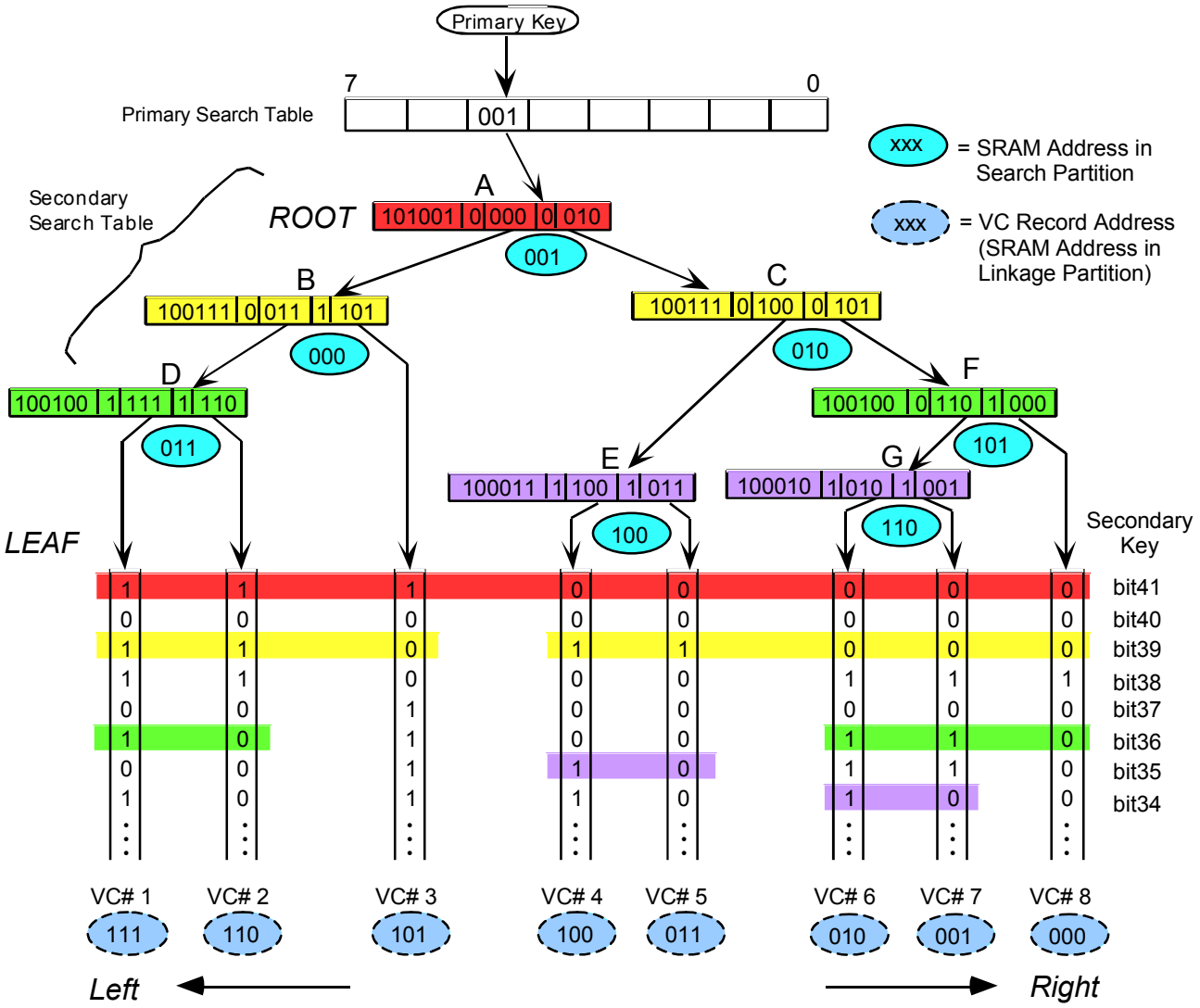
Field	Value	Effect
Unused[4:0]	0	Set unused bits to 0.
Search_From_IBCIF	0	Not relevant to this example, so leave at default value.

Field	Value	Effect
LPHY[2:0]	5	Length of PHY ID in the Primary Key is 5. In this example it will not matter since all connections have identical PHY IDs. The condition that LPHY + LA <= 17 is met.
LA[4:0]	12	Length of Field A in the Primary Key is of 12 bits. This is chosen to match the length of the VPI field in the routing word.
STARTA[6:0]	63	Field A is extracted from the Routing Word with its MSB located at bit position 63, which is the MSB of the VPI.
LB[3:0]	8	Length of Field B in the Secondary Key is 8. This is chosen so that the 8 LSBs of the VCI will be used.
STARTB[6:0]	43	Field B is extracted from the Routing Word with its MSB located at bit position 43, which is the 8th bit of the VCI.

Figure 58 below illustrates the search setup. The Secondary Search Table entries are shown at each node. For simplicity, leading zeros are not included in each field, and only Field B from the Secondary Search Key is shown. Since this example uses the algorithm of searching from MSB to LSB in the Secondary Search Keys, none of the bits below FieldB will be searched so they do not need to be shown. These entries are arranged such that their vertical positions correspond to which bit is used to make the branching decision.

The 3-bit SRAM addresses are shown for the search table entries and the VC Table Records themselves. The SRAM addresses for the Secondary Search Table entries can be assigned arbitrarily. The SRAM addresses for the VC Table Records can also be assigned arbitrarily. In this case, they are assigned in descending order with the VC#'s.

Figure 58 - Detailed Binary Search Tree Example



Suppose a cell comes in corresponding to VC# 2 in Table 3 above. The search will proceed as follows:

1. The Primary Search Key is extracted from the Routing Word:

   = 0b00000 (PHY ID) + 0x005 (FieldA)

2. The Secondary Search Key is extracted from the Routing Word:

   = 0b0000 (zero padding) + 0b1011000 (FieldB = 0xB0) + 0b000000 (PHYID) + 0x00500B0 (VPI/VCI).

3. The value of the Primary Key addresses the entry in the Primary Search Table at location 0x0005. The Primary Search Entry points to the root node (node A) of the Secondary Search Tree at address 001.

4. The root Secondary Search Table Entry (node A) at address is read. The Selector field indicates that bit 0b101001, or bit 41, of the Secondary Search Key should be examined.

5. Bit 41 is a one, indicating that the left branch should be taken. The left leaf indicator is 0, meaning a leaf has not been found yet. Thus, the left branch address, 000, is used to read the Secondary Search Table entry of the next node, node B.

6. In Secondary Search Entry at node B, the Selector field is 0b100111 meaning bit 39 of the Secondary Search Key should be examined.

7. Bit 39 is a one, indicating that the left branch should be taken. The left leaf indicator is 0, meaning a leaf has not been found yet. Thus, the left branch address, 011, is used to read the Secondary Search Table entry of the next node, node D.

8. In the Secondary Search Entry at node D, the Selector field is 0b100100 meaning bit 36 should be used.

9. Bit 36 is a zero, indicating that the right branch should be taken. The right leaf indicator is 1, meaning a leaf has been found and that the right branch address contains a VC Record Address.

10. The VC Record Address can now be to read the VC Record for this connection and confirm that its information matches the data in the initial search keys. If a match is confirmed then the address is found that points to the correct VC Table Record that corresponds to VC# 2!

NOTES

CONTACTING PMC-SIERRA, INC.

PMC-Sierra, Inc.
105-8555 Baxter Place Burnaby, BC
Canada V5A 4V7

Tel: (604) 415-6000

Fax: (604) 415-6200

Document Information: document@pmc-sierra.com
Corporate Information: info@pmc-sierra.com
Application Information: apps@pmc-sierra.com
(604) 415-4533
Web Site: http://www.pmc-sierra.com

PMC-2001159 (P1) ref PMC-2001159 (P1) Issue date: Dec 2000