

SUN MICROELECTRONICS

# ATM622-s

*User's Manual*  
*Single Chip ATM SAR*

April 1997

SME4050BGA





Copyright © 1997 Sun Microsystems, Inc. All Rights Reserved.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT ANY EXPRESS REPRESENTATIONS OR WARRANTIES. IN ADDITION, SUN MICROSYSTEMS, INC. DISCLAIMS ALL IMPLIED REPRESENTATIONS AND WARRANTIES, INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

This document contains proprietary information of Sun Microsystems, Inc. or under license from third parties. No part of this document may be reproduced in any form or by any means or transferred to any third party without the prior written consent of Sun Microsystems, Inc.

Sun, Sun Microsystems, and the Sun logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The information contained in this document is not designed or intended for use in on-line control of aircraft, air traffic, aircraft navigation or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses.

Printed in the United States of America.



# Contents



<b>Introduction</b>	<b>xi</b>
<b>1. Functional Description</b>	<b>1</b>
1.1 Theory of Operation	1
1.2 Architecture and Major Components	2
1.2.1 SBus IO block	2
1.2.2 ATM - PHY Interface (UTOPIA)	3
1.2.3 TX, RX ATM-SYS:	4
1.2.4 Transmit Chaining Byte Packing	4
1.2.5 Receive Chaining	5
1.2.6 TX, RX Cntl RAM:	5
1.2.7 TX, RX Data Buffer Memory:	6
1.2.8 TX, RX FIFO	7
1.2.9 TX, RX Media & UTOPIA Interface	7
1.3 Zero Copy TCP and hardware Checksum	8
1.3.0.1 TCP Hardware Checksum	8
1.3.0.2 Transmit TCP checksum	8
1.3.0.3 Receive TCP checksum	8
1.3.1 Protocol Header and Data split	9
1.3.1.1 Gather on Transmit	9
1.3.1.2 Scatter (Splitting) on Receive	9
1.4 Data Flow	10
1.4.1 Transmit Data Flow	10
1.4.2 Receive Data Flow	10
1.4.3 User Access	11
1.5 SAR Operation	11
1.5.1 AAL5, AAL3/4 and Null ALL	11
1.5.2 Transmit Rate Control (CBR, VBR and UBR services)	12

1.5.3	Packet segmentation (MPEG over AAL5) .....	16
1.5.4	Receive Reassembly .....	17
1.5.4.1	Receive ATM Cell Header decode .....	17
1.5.4.2	1024 Channel Handling .....	19
1.6	Host and Local Memory Data Management .....	19
1.6.1	Host Memory Data Structure .....	20
1.6.1.1	Transmit Data Descriptor rings .....	20
1.6.1.2	Receive Free Buffer Descriptor rings - targeted vs non-targeted VCs .....	21
1.6.1.3	Transmit and Receive Completion ring .....	22
1.6.2	Local (Off-chip) Buffer Memory Data Structure .....	24
1.6.2.1	Transmit Buffer Memory Data Structure .....	24
1.6.2.2	Receive Buffer Memory Data Structure .....	25
1.6.3	On-chip Control RAM .....	26
1.6.3.1	Transmit Control RAM Components .....	26
1.6.3.2	Receive Control RAM Components .....	27
1.7	Interrupts .....	31
1.7.1	Reduced Interrupts .....	31
1.7.1.1	tx_individual_intr, tx_all_intr .....	31
1.7.1.2	RX_intr_wait .....	31
<b>2.</b>	<b>Programmer's Reference Guide .....</b>	<b>33</b>
2.1	Overview .....	33
2.1.1	Direct Descriptor Rings .....	34
2.1.2	indirect descriptor rings .....	34
2.1.3	Transmit data descriptor ring .....	35
2.1.4	Transmit data flow with descriptor rings .....	36
2.1.5	Targeted vs. non targeted vs. non-IP receive VCIs .....	37
2.1.5.1	Receive data flow for non-targeted (and non-IP) VCs .....	38
2.1.5.2	Receive data flow for targeted VCs .....	38
2.1.6	Addressing descriptor rings .....	39
2.1.7	OAM and GFC cell handling .....	39
2.1.8	Packet segmentation (MPEG over AAL5) .....	40
2.2	Memory Data Structures .....	41
2.2.1	Host Memory Data Structure .....	42
2.2.1.1	Transmit Data Descriptor Rings .....	42
2.2.1.2	Transmit Completion Ring .....	46
2.2.1.3	TX data buffers .....	49
2.2.1.4	Receive Free Buffer Descriptor Ring .....	49
2.2.1.5	Targeted buffer completion information .....	51
2.2.1.6	RX Completion Ring .....	52
2.2.1.7	RX buffers .....	56
2.3	On-chip Control RAM Data Structure .....	56

2.3.1	Transmit DMA Block	57
2.3.1.1	Descriptor ring pointers	57
2.3.1.2	Transmit buffer FIFO pointers & Schedule Queue	58
2.3.1.3	VBR Descriptor Ring Kick	62
2.3.1.4	Tx DMA State	63
2.3.2	Rx Initialization Block	67
2.3.2.1	Descriptor ring pointers	67
2.3.2.2	RX DMA schedule queue entry	68
2.3.2.3	RX VCI-DMA State map	69
2.3.2.4	RX LRU linked list	70
2.3.2.5	RX buckets next pointers	71
2.3.3	Receive DMA Block	72
2.4	Off Chip memory	77
2.4.1	Transmit Memory	78
2.4.1.1	Transmit Bandwidth Group Table	78
2.4.1.2	Transmit buffer memory	79
2.4.2	Receive memory	81
2.4.2.1	Rx bucket memory	81
2.4.2.2	Receive buffer memory management	83
2.4.2.3	RX DMA state	85
2.5	Registers, I/O Commands	85
2.5.1	Dynamic Registers	85
2.5.1.1	Global registers & IO Commands	85
2.5.1.2	RX_LRU and free memory registers	89
2.5.1.3	VCI commands per connection	91
2.5.1.4	Turning on a VCI	93
2.5.1.5	VCI/VPI bit selection	93
2.5.1.6	Bad cell counter	94
2.5.1.7	Interrupt registers	94
2.5.1.8	Tx Status Register	99
2.5.1.9	Tx descriptor kick	100
2.5.1.10	TX_descriptor_ring_size	100
2.5.1.11	Tx Bandwidth Group Table Size, Tx Link Rate and Per-Leaky bucket registers	101
2.5.1.12	DMA schedule registers	103
2.5.2	Addressing the RX RAMs	105
2.5.3	IO Address Map	106
<b>3.</b>	<b>Pin-outs</b>	<b>111</b>
3.1	Overview	111
<b>A.</b>	<b>Appendix</b>	<b>117</b>
	Errata	118

*ATM622-s User's Manual*

*Sun Microelectronics*

*vi*



## List of Figures



Figure 1-1	ATM622-s Internal block diagram and data flow . . . . .	3
Figure 1-2	TX, and RX Control RAM organization . . . . .	6
Figure 1-3	TX, and RX Buffer Memory format . . . . .	7
Figure 1-4	Receive hardware TCP checksum coverage . . . . .	9
Figure 1-5	Transmit CBR BWG Table in buffer memory . . . . .	12
Figure 1-6	Transmit flow and rate control . . . . .	14
Figure 1-7	VBR and CBR association with Leaky Buckets (LB) . . . . .	14
Figure 1-8	Per channel Leaky Bucket Register . . . . .	15
Figure 1-9	Packet segmentation Scheduler . . . . .	16
Figure 1-10	MPEG segmentationPacket . . . . .	17
Figure 1-11	ATM cell format . . . . .	18
Figure 1-12	ATM622-s Data Management partitioning . . . . .	20
Figure 1-13	Host Memory descriptor rings . . . . .	23
Figure 1-14	TX channel buffer format to contain multiple packets . . . . .	25
Figure 1-15	Receive path data flow . . . . .	26
Figure 1-16	RX Coalescing Control RAM Data Structure . . . . .	29
Figure 1-17	Load, Unload Data structure components . . . . .	30
Figure 2-1	Direct descriptor rings . . . . .	34

*ATM622-s User's Manual*

Figure 2-2	Indirect Descriptor rings.....	35
Figure 2-3	Transmit data flow with descriptor rings.....	37
Figure 2-4	MPEG segmentationPacket .....	40
Figure 2-5	ATM622-s memory regions .....	41
Figure 2-6	On chip control RAM regions .....	57
Figure 2-7	Tx DMA Control RAM .....	57
Figure 2-8	Logical Init block regions .....	67
Figure 2-9	TX buffer memory format.....	80
Figure 2-10	All buckets on free list.....	83
Figure 2-11	Cells arrive, buckets are taken from free list and placed on VCI lists.....	84
Figure 2-12	Data DMA to memory, buckets are placed at the head of the free list.....	84
Figure 2-13	VPIbits .....	93
Figure 2-14	Badcell counter .....	94
Figure 3-1	ATM622-s pinouts .....	112

## *List of Tables*

---



Table 1-1	Transmit segmentation rate table for a 622/155 link . . . . .	15
Table 2-1	Thresholds . . . . .	59
Table 2-2	Tx FIFO Size. . . . .	60
Table 2-3	Tx Description Ring Size Recodings . . . . .	101
Table 2-4	Setting the registers . . . . .	103
Table 2-5	Batman IO address Map. . . . .	106
Table 2-6	ATM-SYS Register address Map. . . . .	107
Table 3-1	Pin Out Description . . . . .	113

*ATM622-s User's Manual*

*Sun Microelectronics*

*x*

# *Introduction*

---



## *Overview*

The ATM622-s (SBus ATM Host Interface) is a 388 pin BGA single chip solution that implements ATM (Asynchronous Transfer Mode) Segmentation and Reassembly (SAR) and an ATM network system interface. The device connects to an SBus interface. The ATM622-s complies with the ATM-Forum's baseline document (ATM-Forum UNI 3.0 specification).

The ATM622-s supports 155 Mbps or 622 Mbps operation. Because of this versatility and optimized hardware design, the ATM622-s is the only IC to span workgroup, enterprise and backbone applications including network adapter cards and high performance switch interfaces to support interactive broadband networks.

## **ATM Features**

- Conforms to ATM baseline document (ATM-Forum UNI 3.0).
- Performs AAL5 segmentation and reassembly (SAR) in hardware.
- Offers programmable packet segmentation for MPEG over AAL5.
- AAL3/4 in software and Null AAL in software.
- Up to 1024 receive connections.
- Support for CBR, VBR, and UBR traffic - 127 channels transmit and 128 channels receive simultaneously.

## **Major Functional Components**

### **ATM PHY interface**

The ATM PHY interface supports a byte stream at 20 MHz for 155 Mbps (OC-3) and a 16-bit stream at 40 MHz for 622 Mbps (OC-12). Implementing UTOPIA to support 155/622 Mbps, this interface connects ATM SAR chips and PHY/PMD transceivers from different vendors.

## **ATM SAR**

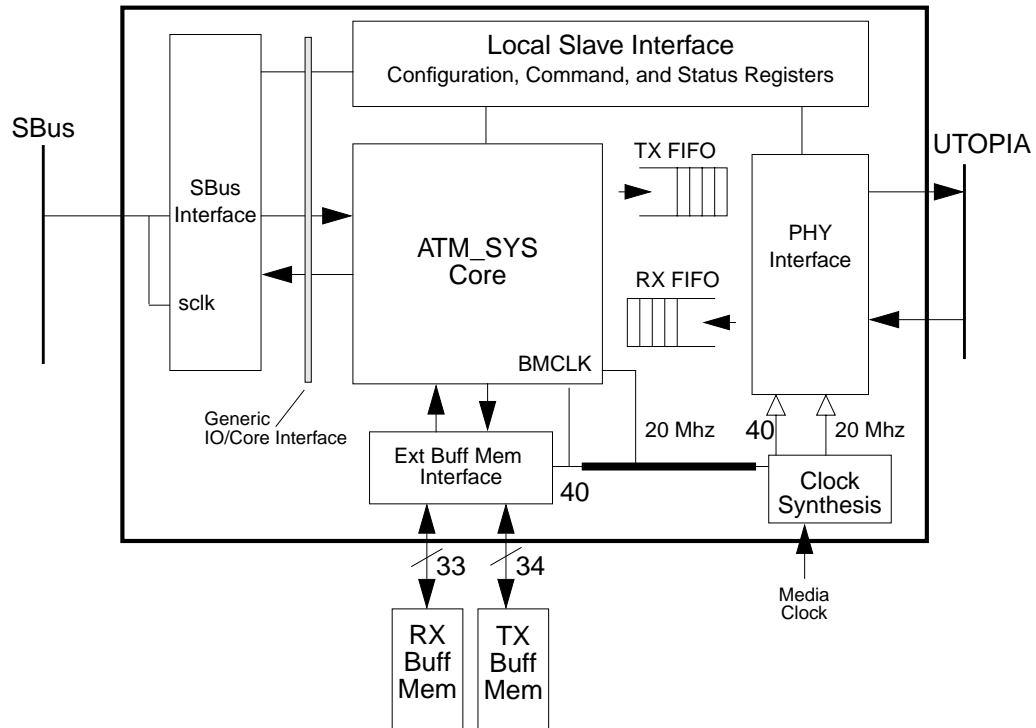
The ATM SAR is responsible for segmenting packet data, posted by the host over transmit channels programmed at different rates, into 53 byte ATM cells. The ATM SAR also performs the reassembly process by reassembling cells to packet data destined for receive channels.

- A packet level interface to the Host on transmit and receive path.
- Host packet management via descriptor rings.
- Posting packets to 128K transmit bandwidth group (TX BWG) independent of VCI/VPI to enable efficient scheduling CBR, UBR, VBR traffic.
- Zero copy TCP hardware hooks (for checksum, data/header splitting) allow direct pipe between media and application data for high performance.
- Programmable hardware hooks reduce interrupts for excellent I/O performance.
- Synchronous SRAM interface for implementing external transmit and receive buffers.
- Minimized IO TLB trashing.
- Rate adaptive RX buffer memory and 256K TX memory provides excellent buffer memory utilization.

## **IO**

The chip implements 64-bit SBus as its I/O interface. The interface between the IO block and system core is designed to be a Generic IO interface (GIO) and can be replaced by other IO buses of interest. The IO block runs at IO clock frequency domain and the rest of the ATM622-s system (ATM and PHY interfaces) operate at 40 MHz. The GIO is the interface that provides slave and master access to the IO system (SBus).

- Two modes of allowed bus transfer sizes to allow DMA either from Host Memory or another SBus card such as a video card.



**Block Diagram**

*ATM622-s User's Manual*





## 1.1 Theory of Operation

The ATM622-s ASIC is a multi-channel intelligent DMA machine on its system side and an ATM SAR connecting to a 53 byte cell stream ATM PHY interface. This PHY interface is also known as UTOPIA. At anytime multiple transmit and receive channels are serviced as virtual channels utilizing a full duplex 155/622 Mbps physical link. Multiple packets, being subscribed to different channels are transferred over SBus to a local buffer memory, external to the ASIC, and later get segmented to ATM 53 byte cells towards the media. The reverse process takes place on the receive path by reassembling the cells in to packets, buffered locally, and later transferring them to Host memory. The ATM622-s ASIC generally hides the details of ATM cells from the user in a packet interface. At the Device Driver level, the user deals with transmit and receive descriptor ring data structures for posting packets and checking status. There are mechanics for taking or delivering data from/to user buffer directly and avoid multiple copies over Host memory bus. For TCP packets, the TCP checksum is performed at the ASIC. The ATM PHY interface is different from typical LAN interfaces (e.g., Ethernet/FDDI) in that the device driver deals with multiple transmit and receive channels rather than single transmit and single receive channels. To support multiple virtual connections at any time requires multiple DMA engines for SAR functionality. To accommodate many DMA engines, the ATM622-s uses a RAM based DMA engine where the DMA states are stored and retrieved per channel DMA opportunity. The DMA RAM, known as Control RAM, is also used to save ATM specific parameters for use by hardware which may be different per channel. The Control RAM is internal to the ATM622-s ASIC. The external RAM is primarily used for local buffering of data. In the transmit case, to support TCP checksum in hardware, the entire packet must be loaded to ext buffer memory while calculating checksum on the fly. The checksum result will then be stuffed to the packet in buffer memory and then segmentation will be enabled. The buffer memory is also

used as multiple transmit and receive FIFOs to map multiple 53 byte cells to multiple 64 byte burst to increase system performance and avoid context switching per 53 byte cell.

## ***1.2 Architecture and Major Components***

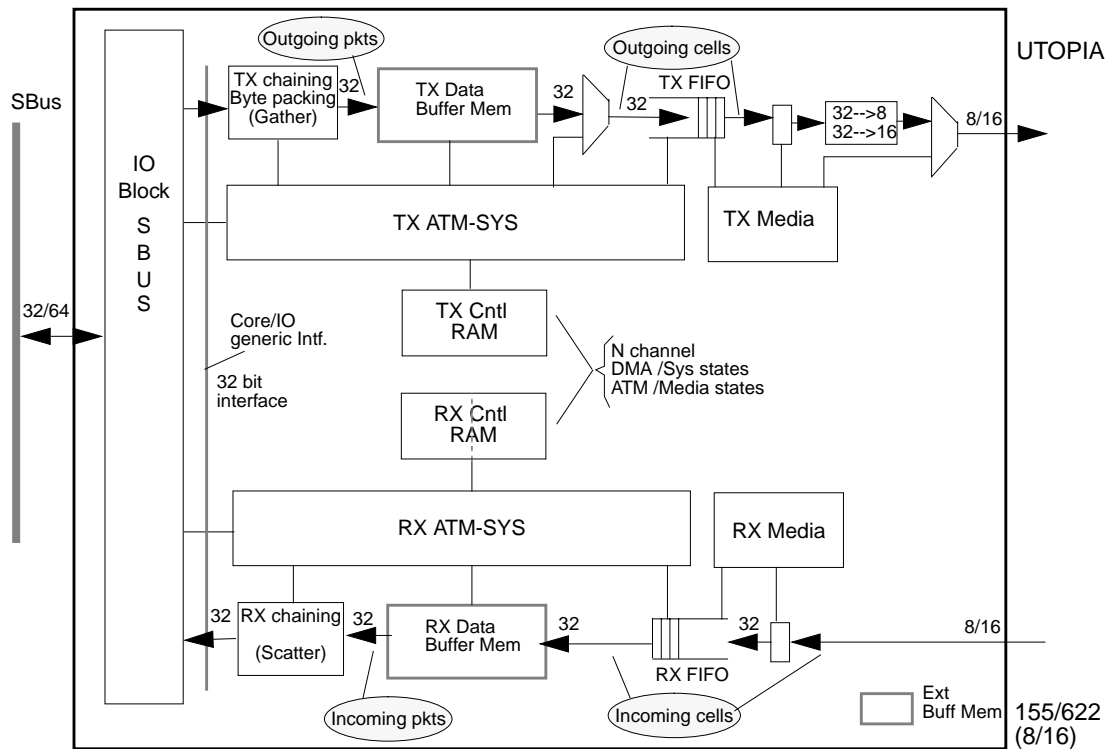
Figure 1-1 shows the major components and data flow internal to the ATM622-s. The transmit and receive data flow are merely symmetrical. The two top level components are ATM-SYS core and the IO block (SBus). The data and control interface between the two blocks are governed by the Core/IO generic interface (GIO). This interface provides two unidirectional 32 bit data paths operating at 40 Mhz. The major components are:

- SBus (IO block).
- UTOPIA interface.
- TX, RX ATM SYS.
- TX, RX Chaining Byte packing.
- TX, RX Cntl RAM.
- TX, RX Data Buffer Memory.
- TX, RX FIFO.
- TX, RX Media & UTOPIA Interface.

### ***1.2.1 SBus IO block***

The IO block in the ATM622-s consists of:

- An up to 64 bit, 64 byte SBus master interface, based on IEEE-P1496 SBus specification.
- A 32 bit SBus slave interface.

**Figure 1-1** ATM622-s Internal block diagram and data flow

### 1.2.2 ATM - PHY Interface (UTOPIA)

There is a proposed definition on 53 byte ATM layer and PHY interface, a *Universal Test & Operations PHY Interface for ATM* (UTOPIA), supported by major computer industries of which Sun is one of the primary contributors. The definition allows a common PHY interface in ATM subsystems across a wide range of speeds and media types. It is intended that ATM SAR solutions from different vendors inter-operate with PHY solutions from other vendors to enable a multi-source environment. The interface defines an eight bit data path running at 20 Mhz for 155 Mbps interface. For 622 Mbps interface, it is a 16 bit data path at 40Mhz. There is a start of cell indicator to establish the 53 byte cell boundary. UTOPIA specification should be referenced for more details.

### 1.2.3 TX, RX ATM-SYS:

TX ATM-SYS is responsible for Host memory packet interface from one side and packet to cell segmentation from the other side. The system and ATM functionality is partitioned via the ext transmit buffer memory. The RX ATM-SYS serves the similar functionality via the ext receive buffer memory. It reassembles the cells in RX buffer memory and transfers them via packet interface to Host memory.

The TX ATM-SYS is responsible for:

- Host memory data structure via descriptor rings including hooks for zero copy TCP.
- On-fly TCP checksum calculation while data being written to buffer memory.
- DMA capability per transmit DMA channel.
- Packet to Cell segmentation per programmed AAL via ext tx buffer memory.
- AAL5 CRC.
- Traffic rate control for CBR, VBR, and UBR channels governed by overall link rate control.

The RX ATM-SYS is responsible for:

- Host memory data structure via descriptor rings including hooks for zero copy TCP.
- On-fly TCP checksum calculation while data being written to buffer memory.
- AAL5 CRC.
- DMA capability per receive DMA channel.
- Cell to packet reassembly via ext RX buffer memory.
- Special handling of received F4/F5 OAM cells.
- DMA state Cache controller for up to 128 receive channels.

### 1.2.4 Transmit Chaining Byte Packing

The transmit chaining byte packing (see Figure 1-1) is done, on-fly, during packet data transfer between the IO block and ATM-SYS core. The transmit byte packing function stems from the 'Gather' functionality of transmit buffer chaining and byte alignment. This module does the proper byte alignment to ensure that data bytes at multiple buffers boundary, belonging to one packet, are packed sequentially in one segment in transmit buffer memory.

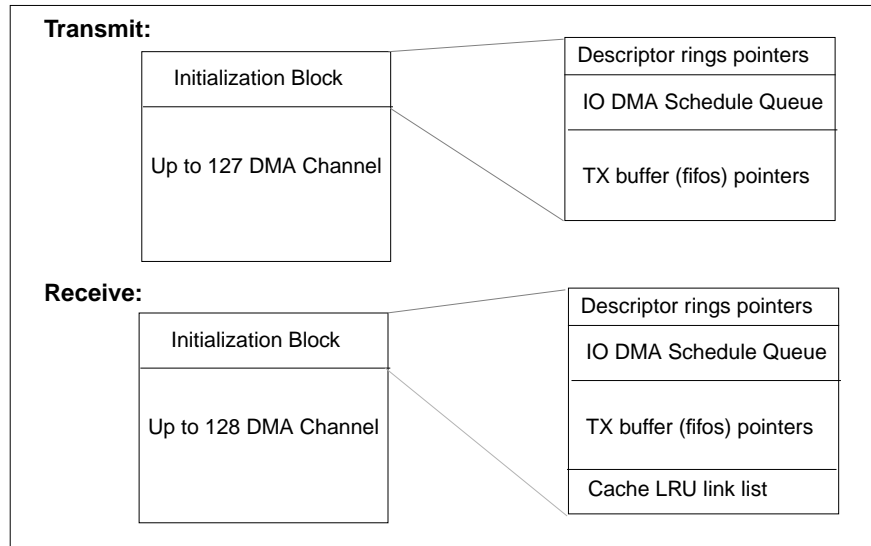
### 1.2.5 Receive Chaining

There is no byte packing in the receive path but there is the support for buffer chaining. Part of receive buffer chaining benefit is used for data and protocol header splitting. For header splitting it is assumed that the header is multiple of 32 bits.

### 1.2.6 TX, RX Cntl RAM:

The two separate control RAMs are primarily used to hold DMA state for multiple transmit and receive channels. Each DMA block (one DMA block per channel) also holds some programmable states for ATM and system specifics. Part of control RAM is used as Initialization block (init block) to hold pointers to descriptor ring in Host memory, FIFO pointers block to contain FIFO parameters for multiple receive and transmit buffers arranged as FIFOs, transmit and receive DMA scheduler queue (fifo) for selecting the next DMA activity for a particular transmit or receive channels. and a LRU link list for scheduling the channel to flush for receive cache controller. The receive control RAM also holds a mirror image of the 1024 DMA channels which reside in Host memory. Out of the 1024 channel, 128 channels are cached internal to the ASIC in the RX control RAM. Any DMA channel flushing and cache update is done over the RX buffer memory. It is perceived that the flushing will be infrequent.

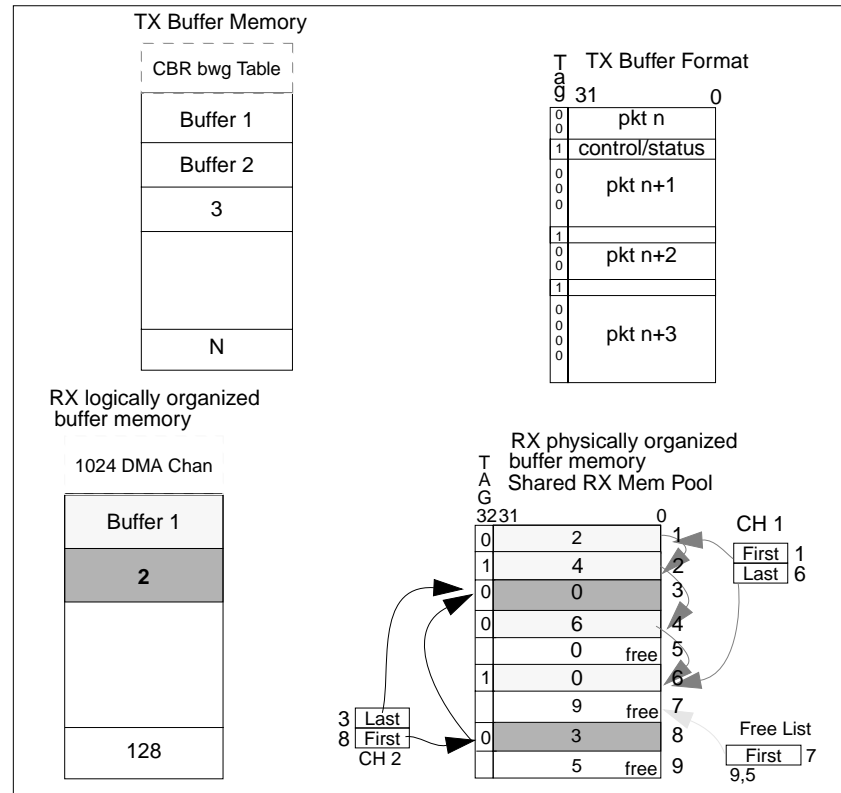
The Control RAM also provides a look-ahead mechanics for TX and RX ATM-SYS DMA engines to ensure that the DMA parameters are known to invoke the next series of IO activity. While bursting data over the IO bus, the control RAM is accessed to stage the next DMA parameters for back-to-back DMA bursts over the IO bus from multiple DMA channels. Figure 1-2 depicts the transmit and receive control RAM organization.

**Figure 1-2** TX, and RX Control RAM organization

### 1.2.7 TX, RX Data Buffer Memory:

There are two separate 33 bit (including one bit of tag) SSRAM for external buffer memory. Each memory supports burst transfers operating at 40 Mhz. Each transmit and receive memory has a bandwidth of 1.2- Gbps out of which half of the bandwidth is used for system packet traffic and the other half for media 48 byte payload cell traffic. The majority of traffic pattern over the memory will be in bursts to obtain maximum efficiency of the bandwidth. However, there will be some single read/write random access to the memory by the Host. The external transmit data memory is segmented to multiple buffers for outgoing packets. There can be multiple packets in each buffer. Each buffer corresponds to one transmit channel. The multiple packet per buffer functionality is done via a tag bit for transmit buffer memory. The buffer memory in reality is 33 bit wide out of which one bit will be used as a tag bit. The receive buffer memory is organized differently than transmit buffer memory. The receive buffer memory is a shared memory pool for all receive channels. The receive memory has been segmented to 48 byte buckets where each bucket holds the 48 byte ATM payload. There is a per channel bucket pointer list (fifo) which holds the order of cells arrived per channel and the index to the cells. This per channel bucket list is used for transferring partial receive packets to host memory. Figure 1-3 shows the top level diagram for RX, and TX buffer memory format.

Figure 1-3 TX, and RX Buffer Memory format



### 1.2.8 TX, RX FIFO

There are separate transmit and receive fifos each holding multiple cells. The fifo is a staging pipe line between the SAR engine and UTOPIA interface. Each cell takes 13 words in the fifo. The transmit fifo size =  $18 \times 33$  and receive fifo =  $70 \times 33$ . The receive fifo is large enough to compensate for buffer memory latency during receive data, and channel flush.

### 1.2.9 TX, RX Media & UTOPIA Interface

The TX and RX Media module are responsible to align cells, being transmitted or received, to the UTOPIA interface. This interface is intended to be a universal interface such that multiple SAR and PHY technologies from different vendors can inter-operate. The interface implies that the PHY module, sitting on the other side of the UTOPIA, has one fifo on transmit path and one on receive. The flags of the

implied fifos are used by the TX and RX media for pushing, and popping data (in 8 bit for 155 Mbps, or 16 for 622 interface) to/from the UTOPIA transmit and receive port respectively.

### ***1.3 Zero Copy TCP and Hardware Checksum***

It is desired that the ATM622-s deals with application data directly via user buffers and avoid multiple data copies flowing through kernel memory. The ATM622-s provides hooks for the kernel to have a direct pipe between the media and the user buffer. The zero copy hardware hooks are:

- TCP/UDP checksum in hardware.
- Protocol header and user data Split. Header to/from kernel, data to/from user buffer.
- Transmit Gather and Receive Scatter function.
- Transmit and Receive buffer chaining.

#### ***1.3.0.1 TCP Hardware Checksum***

The TCP (or UDP) checksum is done in hardware on transmit and receive path. The TCP calculation is done on fly during DMA data transfer from Host memory to buffer memory on the transmit path. For the receive path, the TCP calculation is done during cell transfers to receive buffer memory.

#### ***1.3.0.2 Transmit TCP checksum***

Each transmit TCP channel can optionally program the start offset for calculating checksum and a stuff offset to stuff the 16 bit checksum. The TCP checksum field in the TCP packet is initialized by the software to compensate for the fact that hardware calculates the checksum over the actual IP header rather than pseudo-IP header.

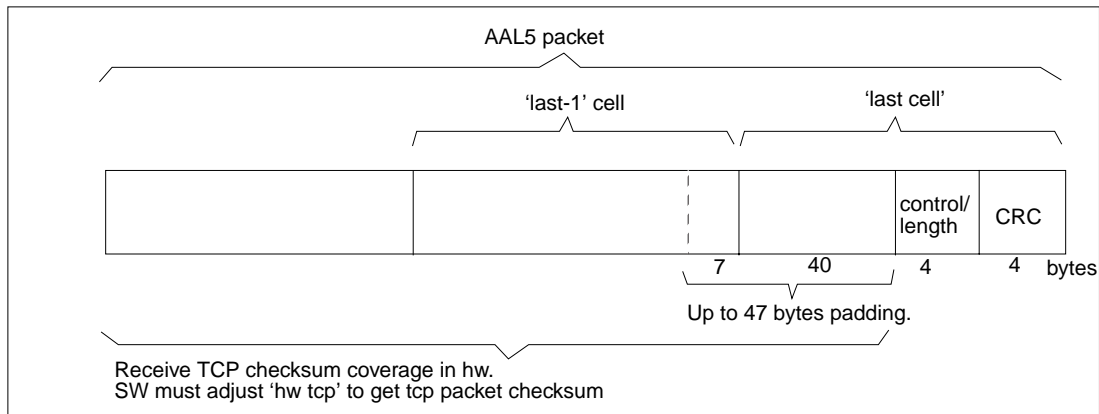
#### ***1.3.0.3 Receive TCP checksum***

The ATM622-s hardware optionally calculates TCP checksum over the entire AAL5 packet and reports it to the software via descriptor. The software must compensate for the additional bytes that have been included in TCP checksum by subtracting them. The TCP checksum calculation starts from the first byte of AAL5 packet until byte 40 of the last cell of the packet. TCP checksum does not



cover the AAL5 tail (control/length field and CRC). Since AAL5 packet may have up to 47 byte padding, the software must compensate for additional bytes (pads) that may have been included in TCP calculation. See Figure 1-4.

**Figure 1-4** Receive hardware TCP checksum coverage



## 1.3.1 Protocol Header and Data split

### 1.3.1.1 Gather on Transmit

The transmit path supports a gather function which consists of gathering, per packet, the protocol header from kernel memory and data from one or more buffers (buffer chaining) in user buffer. The hardware packs the protocol and header in the external off-chip buffer memory.

### 1.3.1.2 Scatter (Splitting) on Receive.

The splitting is also supported in the receive path by a programmable header length field per receive channel. The protocol header is transferred to Kernel memory and data is transferred to kernel memory or *targeted user buffer* (optional per channel). The data transferred to kernel memory may be *page renamed* to user buffer if starts at page boundary.

## 1.4 Data Flow

There are four ports for data to enter/leave the chip. They are:

- ATM/PHY interface for 53 byte cell transfer.
- External buffer memory for 48 byte cell payload and partial packet transfer.
- System IO for partial packet transfer and descriptor/control parameters via DVMA.
- Eprom and PHY slave accesses.

There are also slave cycles for accessing the chip internal registers.

### 1.4.1 Transmit Data Flow

Packets are posted to hardware via descriptor ring data structure in Host memory. There are multiple transmit channels each being subscribed to different transmit rate. The hardware visits the data buffers, belonging to multiple channels, and transfers them partially in multiple bursts (up to 64 byte burst) to local data buffer memory external to the chip. The transmit buffer memory, consisting of multiple buffers (will also be referred as channels, fifos), will be used as fifos before the data being segmented to cells. Packet is pushed in, via transmit DMA engine, to the fifo, and cell gets popped out via segmentation engine. Multiple 48 byte ATM payload are read off multiple transmit buffers and get pushed to internal ATM622-s TX FIFO. The 4 byte ATM cell header (excluding HEC) is pushed to TX FIFO prior to 48 byte payload coming off transmit buffer. There will be a total of 13 words, per cell, pushed in to the TX FIFO. The TX Media module, as shown in Figure 1-1, pops out cells from the TX FIFO and provides the 155/622 streaming to the UTOPIA interface. The TX Media module stuffs the HEC as the 5th byte of the header.

### 1.4.2 Receive Data Flow

Cells arrive at the UTOPIA interface in octet (155 Mbps interface), or 16 bit streams (622 interface). The RX Media, optionally checks for correct HEC before pushing cells to RX FIFO, and will drop cells with incorrect HEC. The cell stream is packed to 32 bit words and pushed in to the RX FIFO in 13 words. The incoming cells, stored in RX FIFO are popped out via receive reassembly engine and routed to the corresponding receive buffer in ext receive data buffer memory. The receive data buffer memory is managed via the 128 cached channels to cover a total of 1024 active VC stream. Once there are enough cells (packed 48 byte cell payload) in the corresponding receive buffer, data is DMAed in multiple bursts

(up to 64 byte burst), via receive DMA engine, to receive buffers in Host memory. Note that only 48 byte payload of the cell is pushed in (not the header) to the receive buffers.

### 1.4.3 User Access

The software accesses the chip, ATM622-s, during the initialization to program registers, bits in the transmit and receive control RAM and during the normal operation for servicing interrupts or tearing down or setting up a connection. There are also slave accesses to EPROM and PHY module.

## 1.5 SAR Operation

SAR functionality is the heart of ATM622-s operation governed by the system and IO interface for a friendly packet interface to the device driver, and the ATM/PHY interface, UTOPIA, for a cell interface. The rest of this section describes the details of SAR.

### 1.5.1 AAL5, AAL3/4 and Null AAL

The ATM622-s primarily supports AAL5 in hardware and has hooks for other AAL to be implemented in software. Other AALs being done in software are solely for limited functionality and are not recommended for high performance packet transfers. AAL5 will be the primary high performance engine for the ATM packet interface.

On the transmit path, the hardware segments non-AAL5 channels just like AAL5 channels by segmenting packets to 48 byte payloads. For AAL3/4 or other AALs, specific AAL encoding within the 48 byte payload must be done in software. For non-AAL5, the hardware does not stuff the AAL5 tail (cntl/length and CRC) to the last cell of the packet. It neither generates nor stuffs the 10 bit CRC for AAL3/4.

On the receive path, the hardware assembles non-AAL5 cells by reassembling 48 byte payloads. For AAL3/4, the hardware detects the end of the packet (by decoding byte 6, bit 6 of every cell). For other non-AAL5 cells (null AAL), the end of packet is recognized by the number of received cells for the corresponding channel programmed by software.

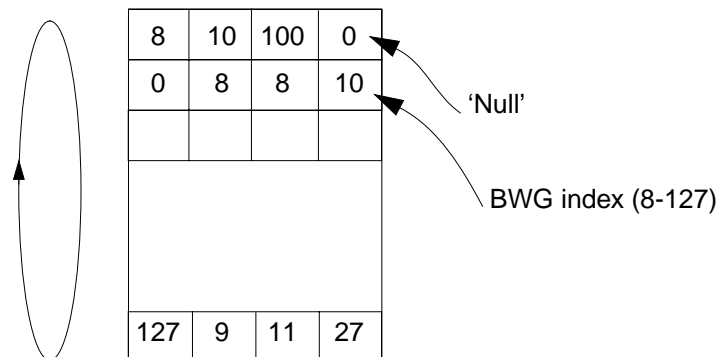
### 1.5.2 Transmit Rate Control (CBR, VBR and UBR services)

The ATM622-s supports three classes of services, Constant Bit Rate (CBR), Variable Bit Rate (VBR), and Unspecified available Bit Rate (UBR). While segmenting packet to cells, the CBR channels have higher priority over VBR channels and VBRs have priority over UBR channels. Each of the three services uses different algorithm for rate control. There are up to 127 transmit descriptor rings out of which 8 are hard-wired to 8 VBR/UBR service and the rest are hard-wired to CBR service. One or more VCs could be subscribed to each of the 8 VBR/UBR transmit descriptor ring and only one VC should be assigned per CBR descriptor ring.

#### CBR Service:

There is a Bandwidth Group Table (BWG) in ext transmit buffer memory (Figure 1-5) that is programmed by software for supporting one or more (up to 120 channels) CBR channels. There are 4 entries (each entry is 7 bits) per memory location and the table size is programmable. The transmit segmentation engine always walks through the BWG table, at every cell boundary decide which channel to segment next. A '0' entry in the BWG table indicates the corresponding CBR channel is quiescent where another VBR channel or UBR channel may be segmented instead.

**Figure 1-5** Transmit CBR BWG Table in buffer memory



#### VBR/UBR Service:

After CBR, the next high priority service is VBR. Each VBR channel, by definition, is associated with a dedicated LB. There are three parameters associated with the LB: Peak rate, Sustain rate (or average rate), and Maximum burst size while segmenting at peak rate. The VBR services are governed by these three parameters. There are 8 LBs out of which 4 are assigned for high priority VBR service and the

rest as low priority VBR service or may be labeled as 4 UBR service whose rates are programmed via LBs. (The LBs peak rate parameter may be programmed as link rate).

The CBR services are governed only by the peak rate control. The ATM622-s supports up to 8 VBR channels and 120 CBR channels. Each VBR channel can also be used as CBR channel. Each channel is configured as a wrap-around descriptor ring where packets (independent of VCs) are queued sequentially. 8 VBR channels correspond to 8 transmit descriptor ring in Host memory. Similarly, 127 CBR channels correspond to 127 descriptor ring in Host memory.

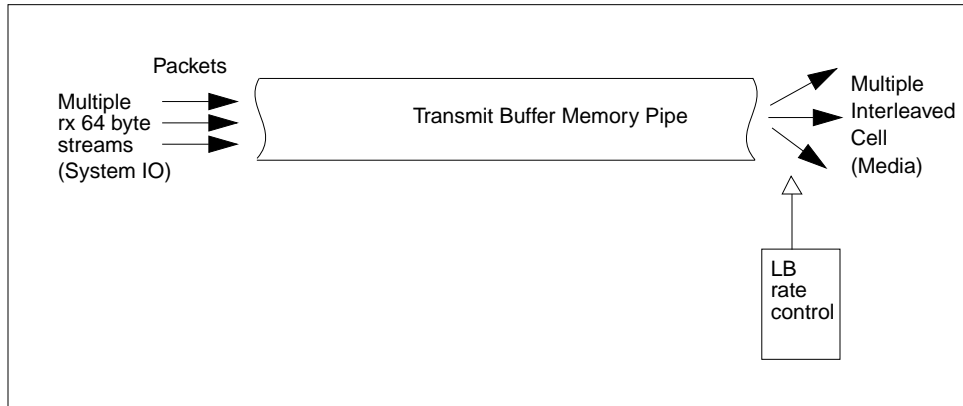
### *Priority for CBR, and VBR/UBR services.*

CBR channels always have higher priority over the VBR/UBR channels for transmission. The transmit engine walks through the CBR BWG table and segments the CBR cells in the order specified. If the CBR BWG table entry indicates a 'null' entry or a CBR channel which has no data to segment, the TX engine segments a cell from the high priority VBR channels (up to 4 channels). If there are no high priority VBR channels which contain data then the low priority VBR channels (also known as UBR channels) are visited. If there is no data to segment from UBR channels either then the particular cell boundary is considered null and no cell is pushed to TX-FIFO within this cell boundary. The priority decision to segment a CBR, VBR, or a UBR channel is done per cell time unit. This guarantees that the link is always utilized (being able to send back-to-back cells at link rate) as long as there are data to segment from any of the CBR, VBR and UBR channels.

The channels within the same VBR or UBR priority are serviced in a round-robin. The overall high and low priority bandwidth programmed can exceed the link rate (155 or 622 Mbps). The over-subscription is used to utilize the link with low priority VBR (UBR) channels when there is no packet to segment on high priority channels.

Buffer memory can be viewed as a pipe whose data flow is controlled at the end of the pipe where cells are segmented. As a result, packets flow through the pipe in the same order as cells leave the pipe (see Figure 1-6).

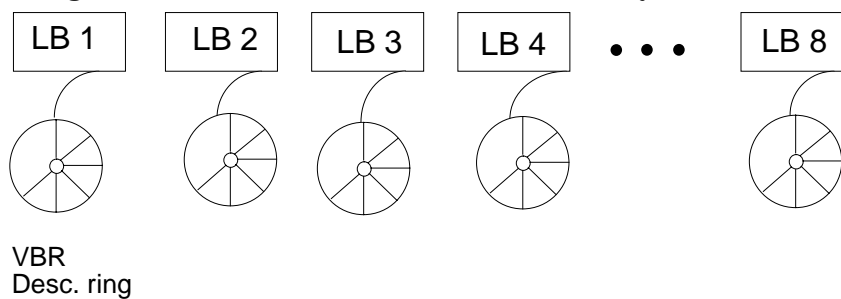
**Figure 1-6** Transmit flow and rate control



*Leaky Bucket Parameters (Peak rate, Sustain rate, and Max burst size)*

Each LB represents three parameters to support both peak and sustain rate. The sustain rate is programmed in fraction of peak rate. The programmable maximum burst size also known as credit bank will limit the number of cells being transmitted at peak rate. ALL three parameters of LB (peak rate, sustain rate, and maximum burst size) must be programmed per VBR channel (For UBR, only the peak rate is programmed to link rate). The sustain rate is a fraction of peak rate in increment of  $1/n$  ( $n=1-16$ ). The maximum burst size is in multiple of 4 cells and applies to VBR channels only.

**Figure 1-7** VBR and CBR association with Leaky Buckets (LB)



**Table 1-1** Transmit segmentation rate table for a 622/155 link

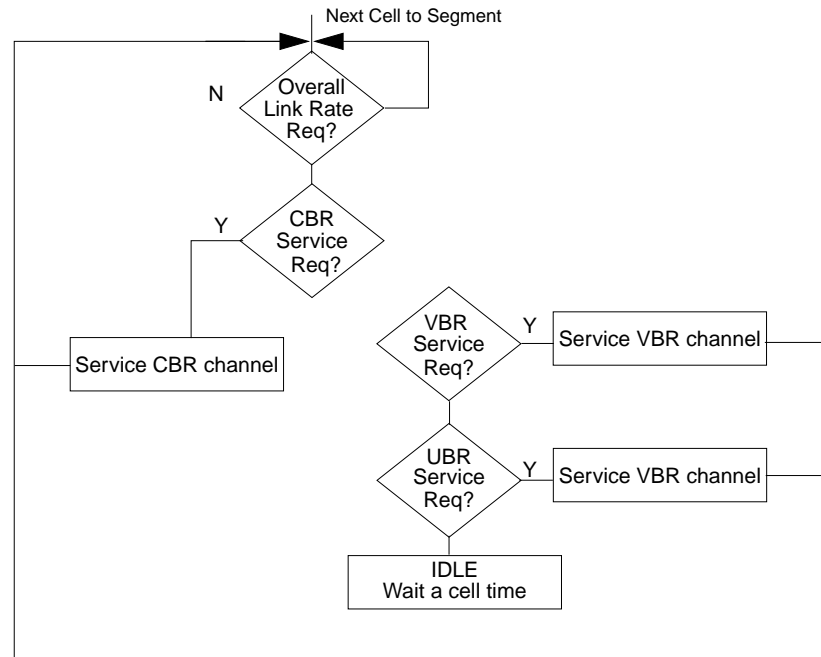
CLK Prescalar	Peak Rate	Sustain Rate (1/n of peak rate, n = 1-16)	Application
16	Link rate-->33 Mbps	Down to 2 Mbps+	ATM domain network 622/155 Links
64	265 Mbps -->8 Mbps (622) Link rate --> 8 Mbps (155)	Down to 500 Kbps	ATM network & FDDI, Fast Ethernet routers
256	66 Mbps -->2 Mbps	Down to 125 Kbps	10 Mbps Ethernet routers
1024	17 Mbps --> 518 Kbps	Down to 32 Kbps+	Voice & Video services

**Figure 1-8** Per channel Leaky Bucket Register

CBR Peak select	Prescalar	Peak rate	Sustain rate	Max Burst Size
1	2	5	4	5 bits

*Maximum Link rate select*

The Leaky Bucket logic in the ATM622-s ATM core does not have (and there is no need for) any knowledge of the maximum physical link rate. The physical link rate could be OC-12, OC-3, or lower rate. There is a programmable register which can be used to set a limit of the maximum link rate. For example, an accumulated peak rate of less than 622 Mbps (e.g., 300 Mbps peak) for all channels can be selected for a 622 physical link. This flexibility may be desired depending on the switching capacity and buffering of a particular switch.

**Figure 1-9** Packet segmentation Scheduler


---

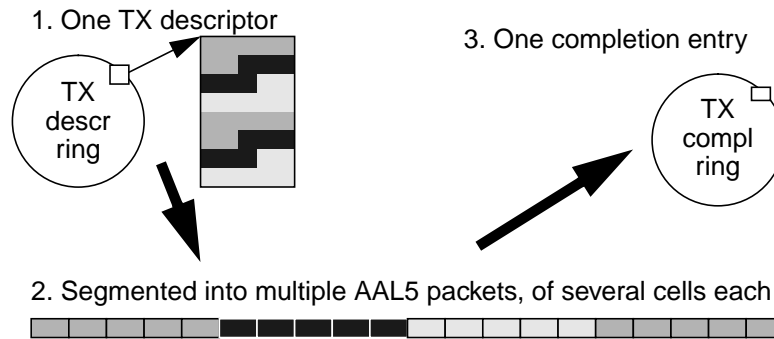
**Note:** Default Overall Link rate is maximum Link rate (155 or 622 Mbps).

---

### 1.5.3 Packet segmentation (MPEG over AAL5)

There is a class of applications which will transmit large sequences of very small packets, where each small packet is the same length. A prime example of this is MPEG video over ATM, which will likely package 1 or 2 MPEG frames (188 or 376 bytes, respectively) in an AAL5 packet. The software overhead of placing each of these tiny packets on the TX descriptor ring and reclaiming from the TX completion ring would be overwhelming. Thus the hardware provides a feature to take a single TX descriptor which points to a large block of host memory, segment this buffer into a large number of small AAL5 packets, and return the entire buffer as a single entry on the TX completion ring.



**Figure 1-10** MPEG segmentation Packet

### 1.5.4 Receive Reassembly

The cell reassembly function is the reverse of what occurs on the transmit path. The cells arrive through RX FIFO and get routed to receive buffer memory before they get DMAed to Host memory. Receive cells from different channels are bunched in external receive buffer memory to provide one or more large burst (up to 64 byte) data transfer to Host memory. Avoiding one cell per channel data transfer, whenever possible, prevents possible system performance degradation particularly with systems with streaming buffer IO interfaces. The receive reassembly supports up to 1024 receive channels (decoding the least significant 10 bits of VCI). Since local buffering and the data structure for 1024 active channel is wasteful and impractical, up to 128 active channels (i.e., up to 128 channels with packets in flight) are serviced at a time. This is accomplished by a fully associative cache mechanism for allocating 128 RX DMA channel dynamically and flushing channels, using LRU algorithm, for more than 128 active channels.

#### 1.5.4.1 Receive ATM Cell Header decode

##### *OAM cell Handling*

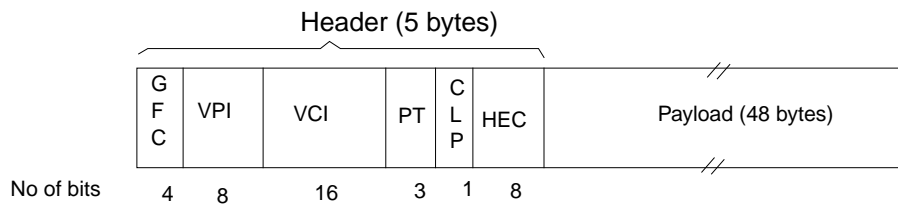
There are two types of OAM cells: F4 flow OAM cells and F5 flow OAM cells (see ATM-forum baseline document). The F4 OAM cells are used to report virtual path connection (VPC) operational information and F5 OAM cells used for virtual channel connection (VCC). The F4 cells have the same VPI value as user data cells but identified by pre-assigned VCI values (VCI of 3, and 4). The F5 cells have the same VPI/VCI values as the user data cells but are identified by pre-as-

signed code point of the Payload Type (PT). There are two unique code points ('100', and '101') to identify management information between segments in the link, and end-to-end.

The ATM622-s treats F4 OAM cells just like any other connection identified by VCI. However, for simplicity, it treats F5 OAM cells belonging to multiple connections as single cell packets all being piped through a pre-assigned RX channel (one out of the 1024 receive channel).

The OAM cells are passed on to the Host as highest priority single cell packet.

**Figure 1-11** ATM cell format



#### *'VPI/VCI' Decode*

The ATM622-s supports up to 1024 receive connections. The VP field is ignored and only the 10 LSB bits of VC field is decoded.

#### *GFC field*

The ATM622-s ignores the value of GFC when decoding the header.

#### *Payload Type (PT) Decode*

PT field is decoded to produce the following signals:

1. The cell is a data cell or F5 OAM cell. F4 OAM cells are handled just like any other data cell.
2. Detect the 'End Of Packet' (EOP) for AAL5 based connections.
3. Detect if cell was involved in a congestion. Keep track of congestion count per connection.

### *1.5.4.2 1024 Channel Handling*

There can be up to 1024 open connections (VC turned on) at any time. Part of receive buffer memory will be used to hold the state information for 1024 channels. Each channel state information holds the DMA state parameters for packet transfers to Host memory, parameters for managing one or more packets in the corresponding receive buffer, system specific options, and ATM specifics options. It may be that the majority or all of 1024 connections are open but not all channels will have simultaneous packets in flight. Therefore, it is assumed that over a small unit of time only a fraction of 1024 channels will have simultaneous packets in flight. It will be over a longer duration that all channels will be involved in receiving packets. The ATM622-s receive engine provides 128 cached channels where each channel has a corresponding channel state as part of the control RAM internal to ASIC. Each of the corresponding cached channels will have an associated data buffer within the external receive buffer memory. As cells arrive for different channels, they get cached in and a dedicated state channel is assigned. when the number of active channel reaches 128, any incoming cell for a new channel is accommodated by flushing the Least Recently Used cached channel and updating the channel with the parameters of the new incoming channel.

## *1.6 Host and Local Memory Data Management*

There are three memory sub-systems associated with the operation of the ATM622-s. They are:

- Host Memory.
- Local Buffer Memory external to the chip.
- Control RAM internal to the chip.

The Host memory holds packets, and the data structure and pointers to packets being transmitted and received. The local ext buffer memory is a temp buffer area for multiple channel SAR functionality. The control RAM holds the DMA states and per channel ATM specifics to handle multiple transmit and receive DMA channel. It also holds the data structure specifics to manage multiple transmit and receive buffers for packets in flight. Figure 1-12 depicts the three memory segments partitioning.

**Figure 1-12** ATM622-s Data Management partitioning

Transmit		Receive	
Control Block		Control Block	ATM622-s Internal Control RAM
DMA state Block (Up to 8 VBR, or 128 CBR)		DMA state Block (Up to 128 cached channels)	
CBR BWG Table		1024 DMA state channels	External Local Data RAM
Multiple Data Buffers		Multiple Data Buffers	
Data Descriptor Rings		Multiple Free buffer ring	Host Memory
Completion Ring		Completion Rings	
Multiple Packets (user or kernel buffers)		Multiple Packets (user or kernel buffers)	

## 1.6.1 Host Memory Data Structure

The transmit and receive data structures are organized via wrap-around descriptor rings. The interaction between software and hardware is managed by an OWN bit. Each descriptor has an OWN bit which is set to 1 if the entry is owned by the hardware or 0 if owned by the software. The owner of the descriptor is responsible for releasing the ownership when it updates the descriptor. Once ownership is released, all words of the descriptor entry must be treated as don't care since the new owner may overwrite any or all of it at any time. The ownership mechanism is somewhat different for transmit data descriptor ring.

### 1.6.1.1 Transmit Data Descriptor rings

There are multiple transmit data descriptor rings, each associated with a pre-programmed bandwidth group. There can be up to 8 rings each being subscribed to a separate VBR Leaky bucket, or up to 120 rings for CBR services. Each descriptor ring will have one or more packets queued for transmission. The allocated band-

width group for the corresponding descriptor ring is shared among packets waiting to be transmitted. Each packet, however, will be transmitted at the programmed average rate (or peak rate for accumulated credit greater than 1). Each descriptor in the ring holds information on a data buffer. Each packet can consist of one or more buffers (transmit Gather function) which would mean one or more descriptors. **For TX data rings OWN bit semaphore is not used.** The software does a kick to hardware every time it posts a packet to the ring. The kick is an IO command whose parameter indicates the tx data ring number (0-126), and the corresponding descriptor number that the software has touched last. The hardware internally keeps track of the last descriptor, per ring, it has processed. When both hardware and software descriptor pointer match, it is an indication that the corresponding ring (channel) has ran out of data. Under this condition, the corresponding channel goes to sleep until it is awakened again by a software kick command. This mechanism prevents hardware from polling descriptor rings when there is no data to send. The similar mechanism is used for completion ring. The software kicks the hw every time an available descriptor is posted to the completion ring. The hw will not poll the Host memory for completion ring descriptor unless there is an available descriptor.

### 1.6.1.2 Receive Free Buffer Descriptor rings-targeted vs. non-targeted VCs

There are two different types of free buffer descriptor rings. Each VC subscribes to one of the two types. There is a common free buffer pool that could be used by multiple VCs. The 2nd type provides per VC dedicated free buffer pool. The VC subscribing to the common free buffer pool is referred to as non-targeted VC and per VC dedicated free buffer pool is known as targeted VC. The data and protocol header splitting is supported by both targeted and non-targeted VC. Targeted VCs will have buffer chaining privilege where the non-targeted VCs will not.

#### *Free non-target (kernel) buffer ring*

There is one common free data ring primarily to store packets for multiple receive channels which have been programmed as non-target channels. Non-target channels are normally used for packets destined to kernel space such as NFS. The free non-target data ring is also used as an auxiliary buffer pool when the individual 'target' channels (channels that are tied directly to user application) run out of free buffers. There are two buffer pointer entries per descriptor. One buffer ptr is used to store the protocol header and the other for data portion of the packet. The header buffers and data buffers are limited to 512 bytes and 64k bytes respectively. Free non-target data ring uses 'OWN' bit for descriptor ownership.

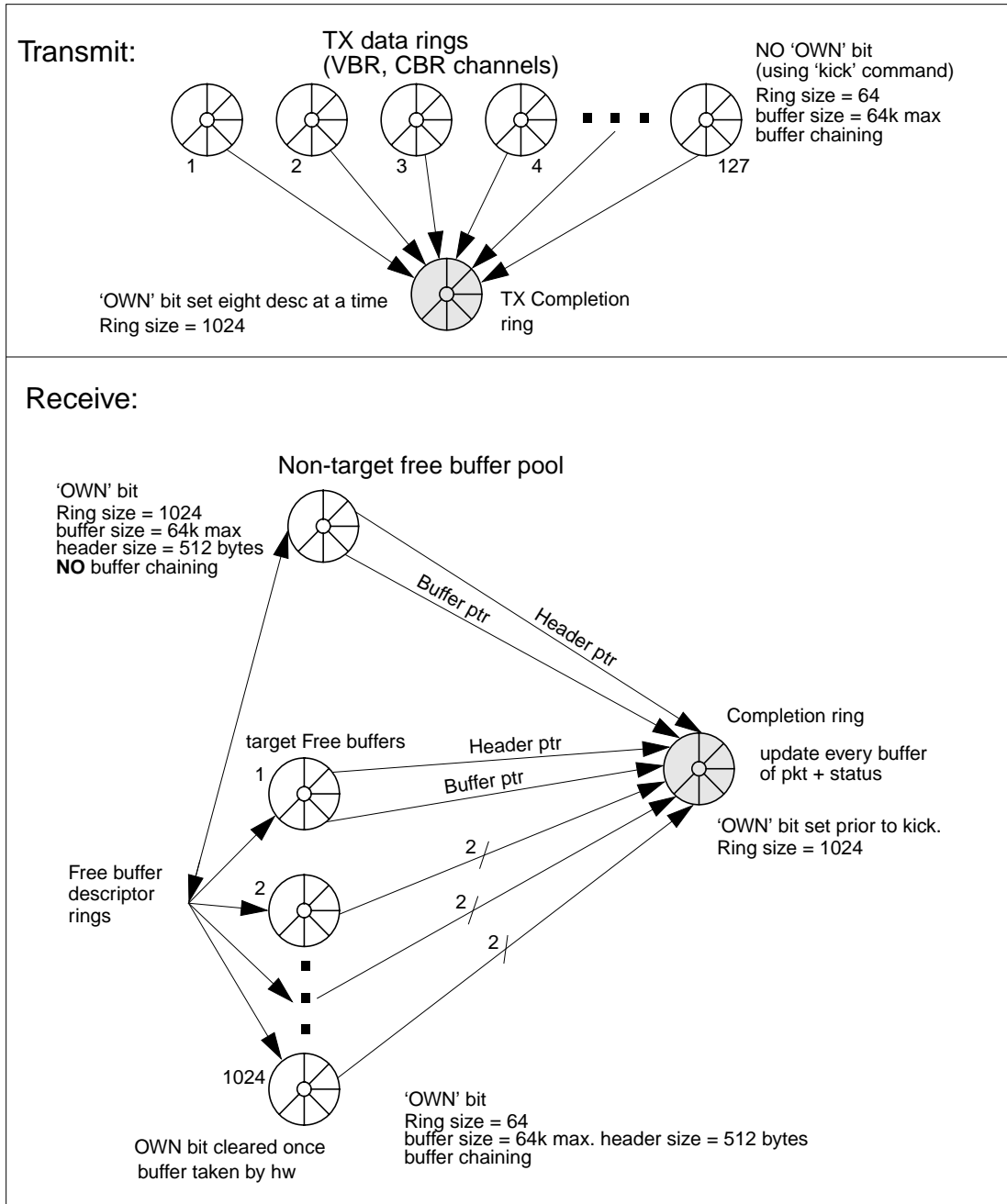
*Free target (user data) buffer ring*

Each receive channel programmed as target channel would have its own free buffer ring. There will be up to 1024 number of free buffer rings for up to 1024 number of target connections. When a particular channel runs out of free buffer and a packet arrives, the hardware will resort to common free buffer pool to get a free buffer. Free target data ring uses 'OWN' bit for descriptor ownership.

***1.6.1.3 Transmit and Receive Completion ring***

There is one transmit and one receive completion ring. The device driver visits, upon interrupt, one data structure for finding the status of multiple channels. On the transmit path, the tx completion ring is used to report the packets that have been transferred to local buffer memory for segmentation. On the receive path, the received packets for multiple channels are reported in single completion ring both for targeted and non-targeted VCs. Transmit and receive completion rings use 'OWN' bit for descriptor ownership but no polling will be done by hw but rather the kick mechanism is used to post descriptors.

**Figure 1-13** Host Memory descriptor rings



## 1.6.2 Local (Off-chip) Buffer Memory Data Structure

The transmit and receive buffer memory managements are handled differently. However, conceptually, they achieve the same functionality in reverse order.

On the transmit path, multiple channels are serviced by transferring packet or partial packet to the corresponding buffer while the current or previous packet for the same channel is being segmented to cell per specified rate control. Or, packets from other channels are being segmented to cells while a different buffer (channel) is being filled for a later segmentation.

On the receive path, cells arrive for multiple channels get buffered in a shared memory pool. The receive data structure manages for each channel the location and the number of cells stored in the shared memory pool. Once there are enough cells per channel, the receive DMA engine packs the cells and transfers them to host memory as partial packet. The process continues until the whole packet is transferred for each channel.

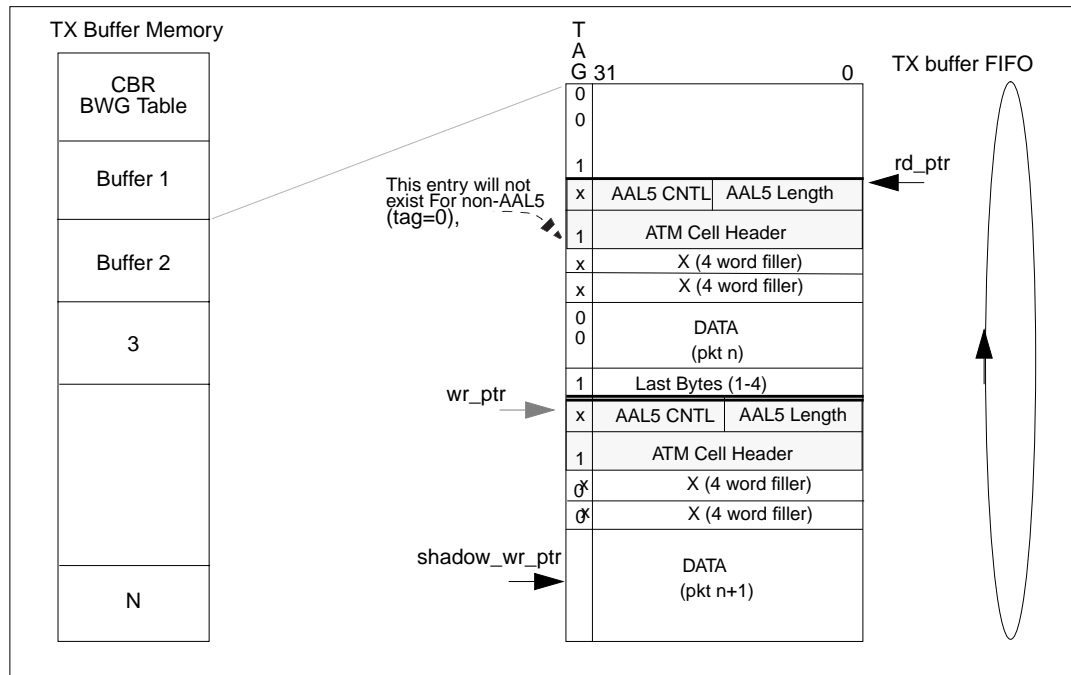
### 1.6.2.1 Transmit Buffer Memory Data Structure

The transmit buffer memory can be segmented up to 127 buffers out of which the first 8 (buffer 0 through 7) are allocated for 8 VBR connections and buffers 8-126 for CBRs. Each channel buffer size can be programmed in multiple of  $2k \times 2^n$  ( $n=1-7$ ) bytes. Packets for each channel flow through the corresponding buffer. Depending on the buffer size, there could be one or more packets in flight in each buffer.

Packets, within each buffer, are separated by tag bits (bit 32 of control RAM). Entries with tag bit set contain ATM 4 byte header and AAL5 tail field namely control and length field. The AAL5 entry will not exist for non-AAL5 packets. For TCP packets the entire packet will be transferred to the corresponding buffer before the cell segmentation starts. This is to calculate, on fly, the TCP checksum and stuff it in the TCP header before transmission starts. In order to inhibit segmentation of TCP packet while being transferred to buffer memory, the *wr\_ptr* visible to segmentation engine (TX Unload) is kept static until the whole packet is written to buffer memory and TCP checksum field is updated. The *shadow\_wr\_ptr* is used as the memory address for write cycles. Once the whole TCP packet is written, the *wr\_ptr* is updated with *shadow\_wr\_ptr*.

For non-TCP packet, the cell segmentation can start (as soon as one cell resides in buffer) while the packet being transferred to the buffer.



**Figure 1-14** TX channel buffer format to contain multiple packets

### 1.6.2.2 Receive Buffer Memory Data Structure

The receive buffer memory primarily is used for buffering the incoming cells. Part of buffer memory is also used to hold the channel state information for 1024 receive channels out of which 128 of the channels will be cached internal to the ASIC.

The data buffer memory is organized as collection of 48 byte buckets. Each bucket holds the payload of an ATM cell. The data buffer memory is a shared memory pool whose 48 byte bucket will be assigned dynamically to the incoming cells for different channels. There will be no pre-assigned buffers for each channel since the traffic pattern and packet size is not predictable. The dynamic bucket allocation assigns free buckets to the channels according to their needs and memory is fully utilized. In the extreme case, one channel can occupy all buckets in the memory pool (In reality, buckets are emptied out, in parallel, by the receive reas-

sembly engine and assigned buckets are freed out unless the system IO introduces a long time latency), or buckets may be assigned to many channels with close to perfect interleave cells.

**Figure 1-15** Receive path data flow

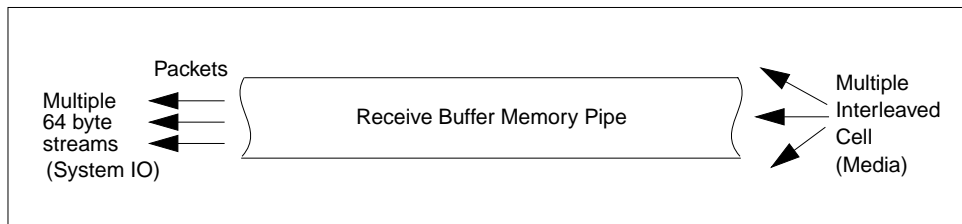


Figure 1-15 shows the receive data flow through receive memory pipe. The receive reassembly engine maps 48 byte cells to multiple maximum burst (up to 64 byte) for optimum IO interface and higher throughput. There are two components to reassembly engine. They are:

- Receive Load engine. Transferring cells to buffer memory.
- Receive Unload engine. Mapping cells to max burst size partial packet.

Figure 1-16 shows the receive control RAM data structure for managing the data for different channels in receive data buffer memory. The data structure is used by both load and unload receive engines.

### 1.6.3 On-chip Control RAM

There are multiple pieces of control RAM used as control data storage for managing multiple transmit and receive channels. Each piece of the RAM may be of different width depending on the data structure it needs to support. The on-chip control components are:

- Initialization block.
- 127 transmit channel block (aka, DMA block).
- 128 receive channel block.
- Transmit buffer memory data structure.
- Receive buffer memory data structure.

#### 1.6.3.1 Transmit Control RAM Components

The data structure components are:

- Initialization block.
- 127 channel state table.
- Unload Schedule Queue.

### 1.6.3.2 Receive Control RAM Components

The data structure components are:

- Initialization block.
- 128 Channel state table.
- Free Channel pool as LRU Link List.
- Free bucket pool and Channel data link list.
- 1024 VCI to 128 channel mapping.
- Unload Schedule Queue.

#### *Channel state table*

There are up to 128 cached channels whose state information are stored as part of on-chip control RAM. Each channel state block would have ATM specific and system specific parameters that can be programmed by software. Some of the entries in the state block are used by hardware for managing the channel activity and hardware control information. Some of each channel state information is also held in the initialization block, part of the on-chip control RAM. By definition, the state channel information are flush-able to external buffer memory for later use (reloaded again upon channel activity), and state information in the initialization block are insensitive to flush process.

#### *Free Channel pool as LRU Link List*

Up to 128 channels can be cached on-chip. The 128 channels are assigned to 1024 incoming channels dynamically in the order of channel activity. It is perceived that over the short unit of time, there will be less than 128 active channels (128 packets in flight) at any time. An LRU link list data structure is used to keep track of the used channel and flush the least recently used channel when needed. As cells arrive for new channels that have not been cached, an available channel is picked up and assigned to the new VC. If all 128 channels are used up, the least recently used channel is flushed out and the channel is assigned to the new VC. The channels which have been scheduled for unload will not be flushed. It is most likely that least recently used channels have no data in buffer memory and will be flushed accordingly.

### *Free bucket pool and Channel data link list*

A shared link list structure is used to keep track of available empty buckets in shared memory pool, and each corresponding channel would use the same link list memory storage for keeping track of the incoming cells destined for a particular VC (see Figure 1-17). The individual channel link list 'First' and 'Last' pointers would reside in channel state table, and a hard register is used to hold the 'First' pointer (pointer to the next available free bucket) for the free bucket pool link list. The channel data link list would have an additional pointer, 'Partial', used by the receive unload engine. The partial pointer is for buckets which have been touched for data removal, but not completely emptied out.

### *1024 VCI to 128 channel mapping*

There is a 1024 to 128 look up table which represents a fully associative cache to keep track of the assigned channels (1-128) and the corresponding VCI (0-1023). Each entry of the table has a tag bit and a field to identify the channel # (1-128). The tag bit indicates the entry in the channel field is valid and channel is active; it also implies that the VC is 'ON'. If the tag bit is zero (channel not active), the entry in the channel field would indicate if the corresponding VC (0-1023) is 'ON' (meaning that connection is open) and whether the channel is programmed as a targeted or non-targeted channel.

### *Unload Schedule Queue*

The 'Unload schedule queue' is shared between the receive load engine and receive unload engine. There is control ram based fifo used by the load engine for scheduling channels to be emptied out once the corresponding channel has bunched enough number of cells in the receive buffer memory (or last cell of packet, EOP cell, is received). The receive unload engine pops entries off the schedule queue to find the next candidate channel for unloading data to Host memory. The unload engine goes to sleep once the unload schedule queue becomes empty. OAM cells (single cell packet) bypass the schedule queue and have the highest priority for being transferred to Host memory.

### *Last Cell of packet format*

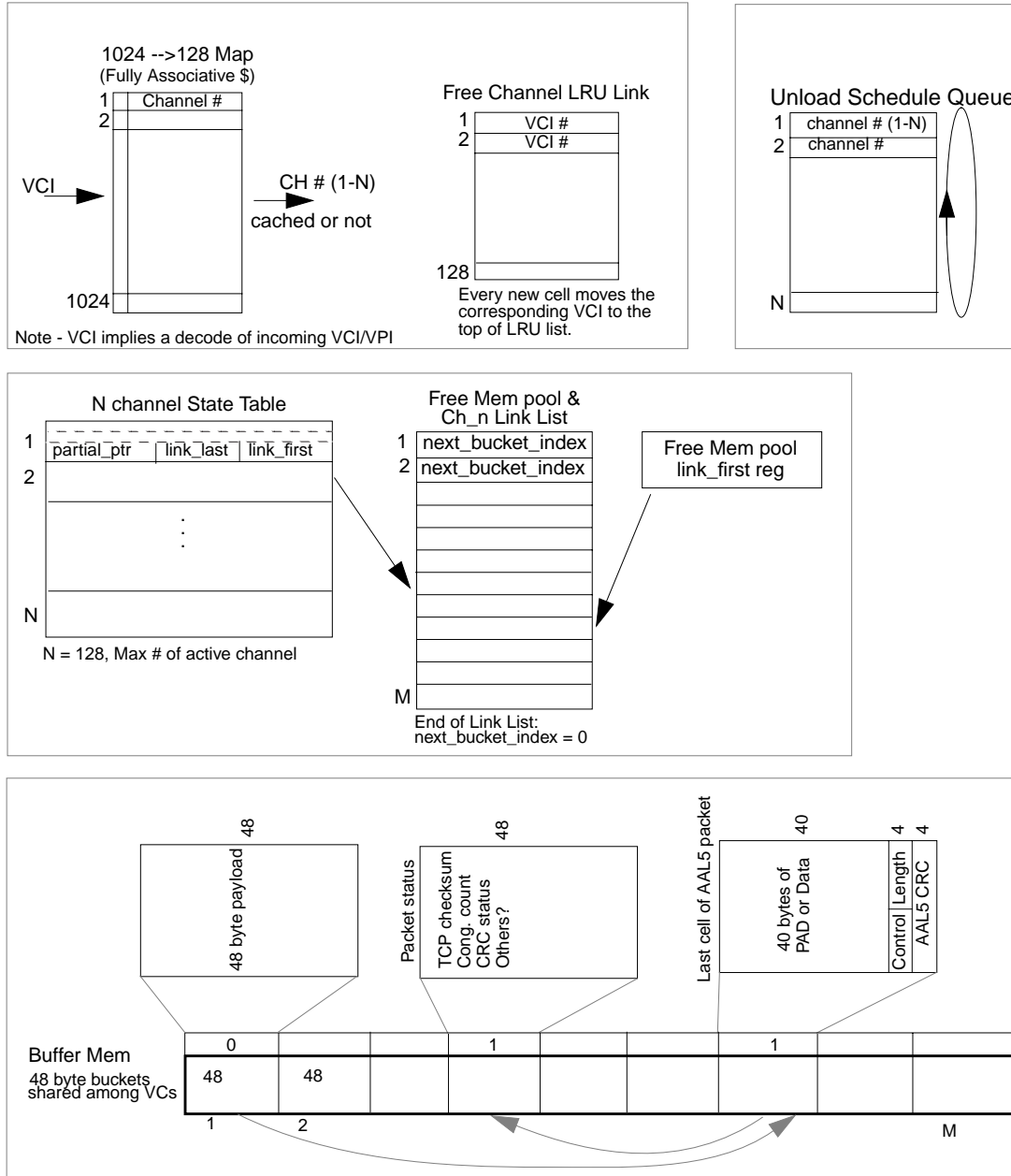
Each bucket in the receive buffer memory is 48 bytes corresponding to ATM 48 byte AAL5 payload. For non-EOP cells, all 48 bytes are considered as data ('last-1' cell of the packet may contain some padding). The EOP cell will have the AAL5 definition format, the last 8 bytes having control/length field and 4 byte CRC. An additional 48 byte bucket is used, per packet, to report status of the received packet (status such as TCP error, CRC error, etc.). See Figure 1-17.

**Figure 1-16** RX Coalescing Control RAM Data Structure

Init Block <i>Initialized by SW. Update by L/U.</i>
1024 -->128 VCI Mapping Table <i>Initialized by SW, Updated by HW L/U.</i>
Free Channel/LRU Pool <i>Initialized by SW, Updated by HW L/U</i>
Free Data Pool & 128 channel data ptr Link List <i>Initialized by SW. Update by L/U.</i>
128 channel state Table <i>Initialized by SW. Update by L/U.</i>
Unload Schedule Queue <i>Initialized by SW. Update by L/U.</i>

L: HW Load engine.  
U: HW Unload engine

**Figure 1-17** Load, Unload Data structure components



## 1.7 Interrupts

Each of the bit in the status register has a corresponding mask bit to enable or disable interrupt upon the status bit being set. There are two sources for causing the interrupt. They are:

1. Interrupt due to the ATM622-s internal status register bit(s) being set.  
Maskable.
2. Interrupt due to an external device to the ATM622-s (ATM PHY module).  
Maskable.

### 1.7.1 Reduced Interrupts

To optimize the transmit and receive packet completion processing, a mechanism is provided to reduce the number of interrupts in cases where there are multiple back-to-back packets being transmitted or received. The per packet interrupt for such scenarios will slow down the system at the Device Driver level. Two separate schemes have been provided to reduce interrupt on transmit and receive.

#### 1.7.1.1 *TX\_individual\_intr, TX\_all\_intr:*

There are two interrupt modes. One interrupt mode provides per packet interrupt capability known as tx\_individual\_intr and the other is tx\_all\_intr for interrupting at last packet posted to a tx descriptor ring. In this mode, one interrupt is generated per descriptor ring upon last packet transfer to buffer memory.

#### 1.7.1.2 *RX\_intr\_wait*

Specifies the number of 20 MHz clock cycles to wait after a receive completion ring entry is posted before giving an interrupt. The effect of this counter is to reduce the maximum interrupt rate; for small packets much less than one interrupt per packet will be given.





## *2.1 Overview*

The ATM622-s programmer's interface is based upon wraparound descriptor rings in host memory. The interaction between software and hardware is managed by an OWN bit. Each descriptor has an OWN bit which is set to 1 if the entry is owned by the hardware or 0 if owned by the software. The owner of the descriptor is responsible for releasing the ownership when it updates the descriptor. Once ownership is released, all words of the descriptor entry must be treated as don't care since the new owner may overwrite any or all of them at any time. There are two types of descriptor rings in the ATM622-s, direct and indirect.

The ATM622-s manages two memory areas: a control memory block on chip, and a buffer memory in an external SSRAM. The control memory contains DMA state information for transmit and receive channels and pointers to data structures in host memory. The buffer memory contains packet data for all transmit and receive channels, and a backing store for RX DMA states not currently cached on chip. In general software must initialize the both memory areas at poweron, but must not interfere thereafter.

The ATM622-s, like most networking cards, signals interrupts to notify the software of important conditions. However, the ATM622-s contains circuitry to vastly reduce the number of interrupts posted compared to previous networking cards. Software must initialize the interrupt generation registers on poweron, and must service interrupts once the chip is in operation.

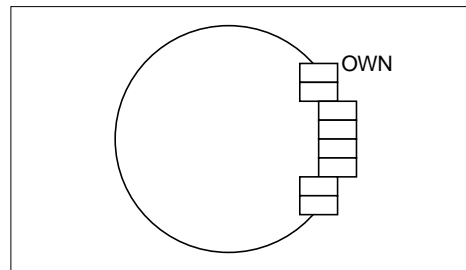
### 2.1.1 Direct Descriptor Rings

Direct descriptor rings contain all of their information in the ring entry itself, rather than through a pointer. The TX and RX completion rings are direct descriptor rings. Direct rings have 1024 entries, where each entry is 64 byte aligned. Thus each direct ring occupies 64 Kbytes of memory. The first entry in the ring must be 64 Kbyte aligned (meaning the address must be of the form 32'hxxxx0000).

When software wishes to give completion descriptors to the hardware it must set the OWN bit in each descriptor, and then write the entry number of the last descriptor to a special hardware 'kick' register. For example, if software is prepared to give TX completion descriptors #0-99 to the hardware it must set all 100 own bits, then write the value 99 to the TX completion kick register. Hardware relies on the kick value to determine which descriptors it owns and can write to; hardware does not read from the ring to see if the own bit is set. Hardware will clear the own bit when it writes to the ring.

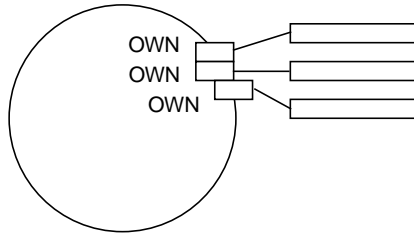
Because a PIO write to the kick register can be expensive, it is recommended that the software wait until it has several descriptors to hand over before giving the kick.

**Figure 2-1** Direct descriptor rings



### 2.1.2 Indirect descriptor rings

Indirect rings contain pointers to other structures, generally a data buffer. The TX data, RX header, and RX data buffer rings are indirect rings. The first entry in the ring must be aligned. Each indirect ring descriptor entry contains an own bit which will be read by the hardware; indirect ring entries may be queued individually.

**Figure 2-2** Indirect Descriptor rings

### 2.1.3 Transmit data descriptor ring

The transmit descriptor ring is a special case of an indirect ring. When a packet is queued to the transmit descriptor ring, the software must write the descriptor entry number to the TX\_kick register for this descriptor on the card. This is done to avoid polling the TX descriptor ring; polling is extremely expensive on most of Sun's platforms. The hardware saves the descriptor entry number, and will transmit packets until it has finished transmitting the descriptor entry number most recently written by the software. The hardware does not write back own bits to the transmit data descriptor ring; software must look at the TX completion ring to determine when a buffer has been transmitted. Thus OWN bits are not used on the transmit descriptor ring.

The TX descriptor ring is treated specially because it is anticipated that packets will be queued in chains, with one entry for the protocol headers (LLC, IP, TCP, UDP, etc), and one entry for the user data. This involves a great deal of overhead per packet, especially if the data field is small. Thus the TX data descriptor ring is treated specially to optimize its performance.

The size of the tx data descriptor ring for CBR channels is 64 entries. The size of the tx data descriptor rings for VBR channels is programmable from 64 to 8192.

---

**Note:** Reread the previous paragraphs. There are no own bits on the TX data descriptor ring. Do not depend on own bits in the TX data descriptor ring.

---

### 2.1.4 Transmit data flow with descriptor rings

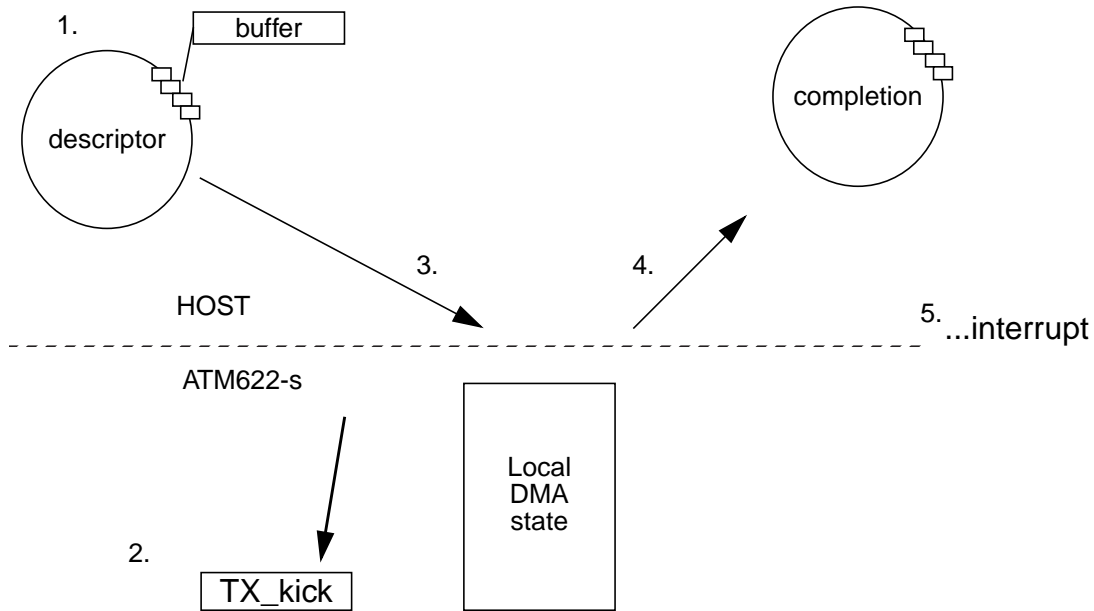
Once the chip is initialized and operating, the general order of operations to transmit a packet is:

1. Software will place descriptors for the packet on the appropriate TX data descriptor ring. See section 2.2.1.1 for details.
2. Software writes the number of the last entry in the TX data descriptor ring to the TX\_descriptor\_kick register on the ATM622-s for that descriptor. See section 2.5.1.7 for details.
3. The ATM622-s will read the entries from the TX data descriptor ring and transmit the packet. It will not write back OWN bits to the TX data descriptor. See section 2.2.1.1 for details.
4. The ATM622-s will write to the completion ring and will generate a tx\_c\_indiv interrupt. This interrupt has an individual mask bit for ease of toggling in software.
5. When the ATM622-s finishes the descriptor from the most recent kick, it will give a tx\_c\_all interrupt. This interrupt also has an individual mask bit for ease of toggling in software.

---

**Note:** The hardware starts from descriptor number 1, not zero. After initializing a DMA state, queue the first packet to descriptor number 1.

---

**Figure 2-3** Transmit data flow with descriptor rings

### 2.1.5 Targeted vs. non targeted vs. non-IP receive VCIs

A particular VCI can be either targeted, non-targeted, or non-IP. A targeted ring has its own buffer ring, where only packets from that VCI will be placed. All non-targeted rings share a common buffer ring in the kernel. All non-IP VCIs share a buffer ring in the kernel, separate from the non-targeted ring. If a packet arrives for a targeted VCI and no buffer is available on the targeted buffer ring, the ATM622-s will abort the VCI to non-targeted and use the non-targeted buffer ring from that point on. There is a command available for software to change the VCI back to targeted. There is no command to switch a VCI from non-IP and back.

Buffer chaining is only allowed on the targeted VCIs. Partial chain information is written back to the buffer descriptor ring, and packet status is written to the RX completion ring at the end of packet. Non-targeted VCIs (and targeted VCIs which aborted) cannot chain buffers.

The purpose of the non-IP ring is to allow those VCIs with raw ATM applications which may use large (64K) packet sizes to have their own buffer ring where a few huge buffers can be placed. Since IP uses 9188 byte packets, the non-targeted ring can be populated with many 9K buffers.

### *2.1.5.1 Receive data flow for non-targeted (and non-IP) VCIs*

The steps involved to set up a non-targeted VCI and receive a packet are shown below. Not all steps are software-visible, they are given here as an aid to debugging.

1. Software initializes a VCI by setting up the words in the ATM622-s external DMA state table and writing to the VCI map (section 3.4.2.3).
2. When a cell arrives, the ATM622-s will take a free bucket from the free bucket list, store the cell data in the bucket, and place it on the linked list for this VCI.
3. When enough data has arrived to do a large DMA write (3 cells), the ATM622-s will read from the RX non-targeted data buffer ring (or the non-IP ring, if set). It picks up both a header and data buffer from the descriptor.
4. The ATM622-s will DMA a software programmable number of bytes into the header buffer to split off the TCP/UDP/IP headers.
5. When additional cells arrive, the ATM622-s will begin to DMA them into the data buffer.
6. When the EOP cell arrives, the ATM622-s will write an entry to the RX completion ring (section 2.2.1.6).
7. The ATM622-s will start a countdown timer, and increment the count of the number of packets received. It will interrupt whenever either counter reaches a threshold, and then clear both counters. See section 2.5.1.7 for details.

### *2.1.5.2 Receive data flow for targeted VCIs*

The steps involved to set up a targeted VCI and receive a packet are shown below. Not all steps are software-visible, they are given here as an aid to debugging.

1. Software initializes a VCI by setting up the words in the ATM622-s external DMA state table and touching the VCI command register (section 2.4.2.3).
2. When a cell arrives, the ATM622-s will take a free bucket from the free bucket list, store the cell data in the bucket, and place it on the linked list for this VCI.

3. When enough data has arrived to do a large DMA write (3 cells), the ATM622-s will read from the RX targeted data buffer ring. It writes back to the targeted buffer ring to clear the own bit and set the chain bit if buffer chaining is involved. If no buffers are available on the targeted ring, go to 8. ATM622-s picks up both a header and data buffer from the descriptor.
4. The ATM622-s will DMA a certain number of bytes into the header buffer to split off the TCP/UDP/IP headers.
5. When additional cells arrive, the ATM622-s will begin to DMA them into the data buffer.
6. When the EOP cell arrives, the ATM622-s will write an entry to the RX completion ring (section 2.2.1.6).
7. The ATM622-s will start a countdown timer, and increment the count of the number of packets received. The ATM622-s will interrupt whenever either counter reaches a threshold, and then clear both timers. See section 2.5.1.7 for details.
8. If no buffers were available on the targeted buffer ring, the ATM622-s will abort to the non-targeted ring, and set the kernel\_abort bit instead. The ATM622-s will from that point on always use the non-targeted ring for this VCI, until a command to switch back to targeted is received.

### *2.1.6 Addressing descriptor rings*

It is recommended that physical addressing be used for the descriptor rings, to avoid thrashing the small IO-TLB on sun4m systems. Direct descriptor rings are 64 K in size, and must be allocated in physically contiguous pages if physical addressing is used. Descriptor rings should be mapped consistent mode on systems with streaming buffers.

### *2.1.7 OAM and GFC cell handling*

There are two types of OAM (signalling) cells, F4 and F5. F5 cells are single cell messages which can arrive on any VC. F4 cells are part of a larger signalling packet (using AAL5) which arrive on VCI 3 or 4.

F5 cells are handled specially by the ATM622-s. VCI 0 represents idle cells (which are discarded), so the ATM622-s uses the DMA state for VCI 0 for F5 OAM cells. Regardless of which VCI the F5 OAM cell arrives on, the ATM622-s will DMA the cell's contents to memory according to the DMA state in VCI 0. VCI 0 must be set

up as a NULL AAL. When an F5 cell arrives, the ATM622-s will go to the buffer ring for VCI 0, DMA the cell's contents to the buffer, and write to the completion ring.

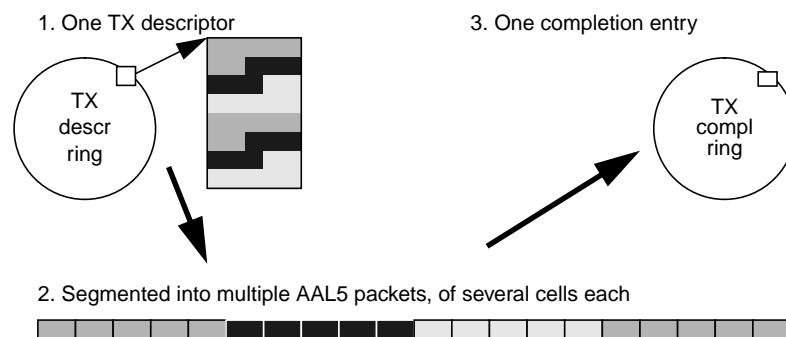
F4 cells on VCI 3 and 4 are treated just like any other connection. VCIs 3 and 4 should be set up as AAL5 connections. They may be either targeted or non-targeted. There are no special restrictions on VCIs 3 and 4 on receive, they are treated exactly like other VCIs by the hardware.

The ATM622-s ignores the GFC field on received cells.

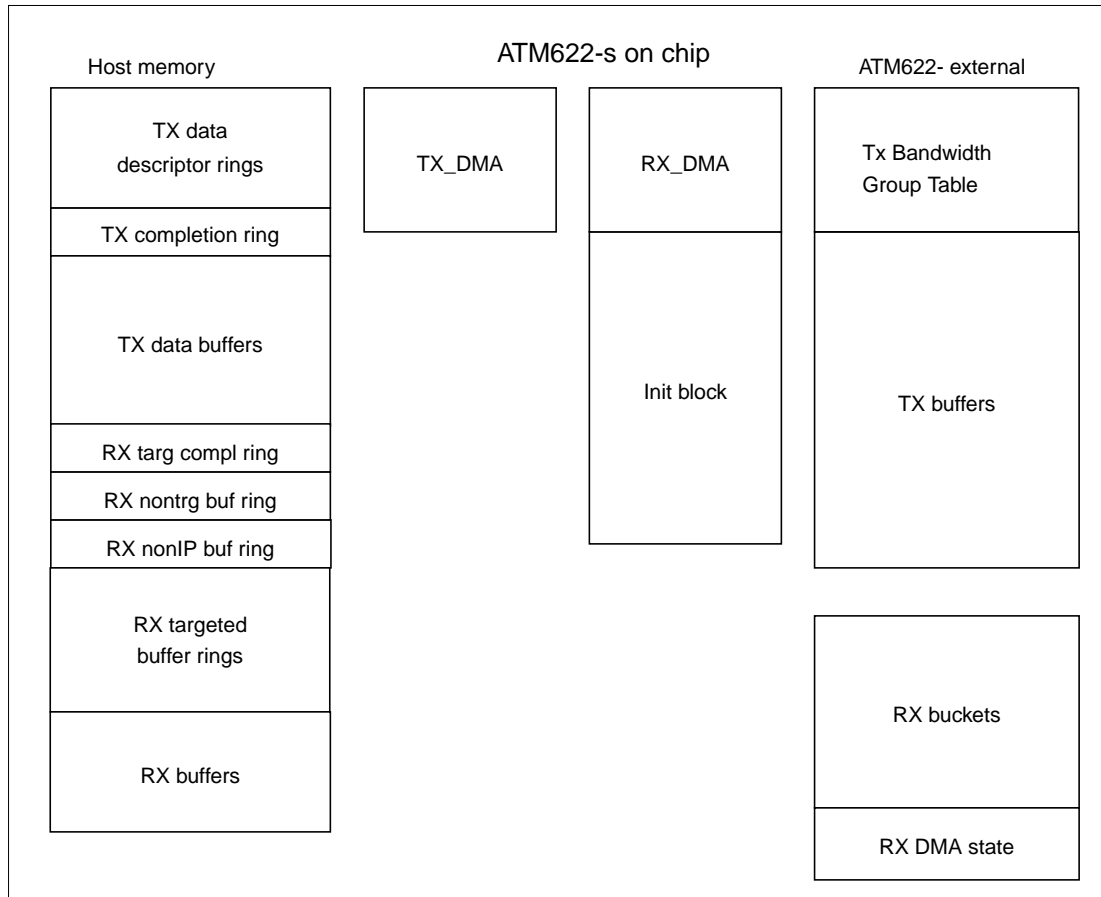
### 2.1.8 Packet segmentation (MPEG over AAL5)

There is a class of applications which will transmit large sequences of very small packets, where each small packet is the same length. A prime example of this is MPEG video over ATM, which will likely package 1 or 2 MPEG frames (188 or 376 bytes, respectively) in an AAL5 packet. The software overhead of placing each of these tiny packets on the TX descriptor ring and reclaiming from the TX completion ring would be overwhelming. Thus the hardware provides a feature to take a single TX descriptor which points to a large block of host memory, segment this buffer into a large number of small AAL5 packets, and return the entire buffer as a single entry on the TX completion ring.

**Figure 2-4** MPEG segmentationPacket





**Figure 2-5** ATM622-s memory regions

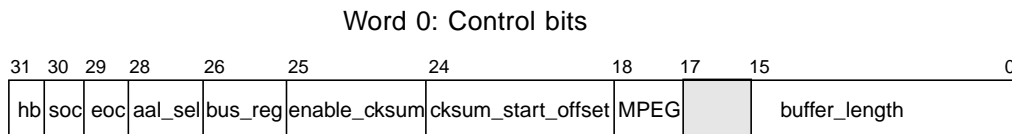
## 2.2 Memory Data Structures

The ATM622-s has two major memory areas, in host memory and local memory (on the card). The local memory is further divided into on-chip and off-chip memory. See Figure 2-5 for an overview.

## 2.2.1 Host Memory Data Structure

### 2.2.1.1 Transmit Data Descriptor Rings

Each transmit queue has a data descriptor ring, where the driver queues up packets to be sent. The transmit data descriptor ring is a special type of indirect ring, see section 2.1.3 for details.



#### *hb (31):*

Hardware bit. There is no own bit, but this bit should always be set to a “1”. This bit is used by the hardware internal to the ATM622-s. The hardware clears this bit **internally** only, to indicate that it is done with this descriptor entry.

#### *soc (30):*

The Start of Chain bit indicates that this is the first buffer in a chain of buffers to be gathered and sent as a single packet. Single buffer packets will have this bit set.

#### *eoc (29):*

The End of Chain bit indicates that this is the last buffer in a chain of buffers to be gathered and sent as a single packet. Single buffer packets will have this bit set.

#### *aal\_sel (28:27):*

This field selects which AAL framing should be used for this packet. The encodings are:

00 AAL5

01 Reserved

10 NULL/Software AAL

11 Reserved

The hardware does all necessary padding and CRC generation for AAL5 packets. Hardware does nothing for Null AAL packets, allowing software to handle all necessary fields. Software must make sure that the packet is an exact multiple of 48 bytes. The reserved encoding is unused at this time. These bits are only meaningful on the first descriptor of a packet, it is ignored on subsequent descriptors.

*bus\_reg (26):*

The ATM622-s contains 2 registers of allowed bus transfer sizes, to represent two different devices in the system (see dynamic registers section). If set, SBus size register 1 will be used, if not set register 2 will be used. Each descriptor in a chain is allowed to use a different bus\_reg setting, for example to DMA the TCP & IP headers from the kernel memory and the packet data from an SBus framegrabber card.

*enable\_cksum (25):*

If set, the ATM622-s will stuff a TCP/UDP checksum calculated starting at cksum\_start\_offset and stuff it at cksum\_stuff\_offset. This bit is only meaningful on the first descriptor of a packet, it is ignored on subsequent descriptors. When this bit is active the entire packet must be written into External Memory before the checksum is completed and since CBR channels are generally allocated less than a packet's size worth of External Buffer Memory this bit should probably not be set for CBR channels.

*cksum\_start\_offset (24:19):*

Gives the number of from the beginning of the packet to skip before beginning the TCP/UDP checksum. Only valid if the enable\_cksum bit is set and only read on the first descriptor of a packet, it is ignored on subsequent descriptors.

*MPEG (18):*

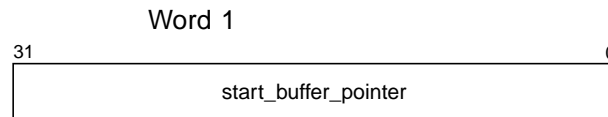
If set, this descriptor should be treated as an MPEG segmentation packet. The buffer length in word 0 is the length of a single MPEG frame, in bytes. The checksum stuff offset field in word 2 is redefined to be the number of MPEG frames in this descriptor. If the MPEG bit is set, the enable\_cksum bit in word 0 must be 0, you cannot use TCP checksumming on MPEG frames. The aal\_sel field must be set to 00, meaning AAL5, if the MPEG bit is set.

*reserved (17:16):*

This field is reserved by the hardware. It should be programmed to all 0s by software.

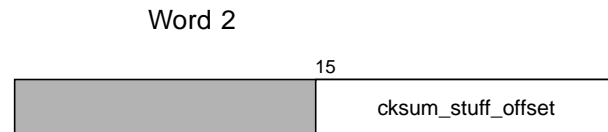
*buffer\_length (15:0):*

Specifies the number of bytes to be transmitted from this buffer. The maximum buffer size is 64 K bytes.



*start\_of\_buffer\_pointer:*

32 bit address of the first byte of data in the buffer. There are no alignment restrictions.



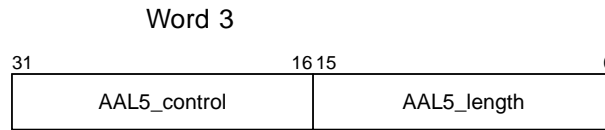
*cksum\_stuff\_offset (15:0):*

Where to stuff the calculated TCP/UDP checksum, programmed as the number of bytes from the first byte of the packet. This field must be half word aligned from the beginning of the packet, only meaningful if the enable\_cksum bit in word 0 is set and is only read on the first descriptor of a packet, it is ignored on subsequent descriptors.



*num\_MPEG\_frames (15:0):*

If the MPEG bit is set in word 0, this word contains the number of MPEG frames to be sent from this descriptor. The buffer\_length field in word 0 contains the length of a single MPEG frame. Thus the host memory buffer pointed to by the start\_buffer\_pointer in word 1 must be num\_frames \* buffer length in size.



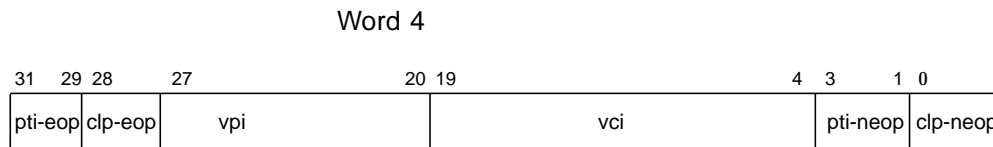
AAL5 specific information. The hardware will take care of padding to place these fields properly in the outgoing packet. For the exact meanings of these fields see the relevant CCITT/ITU standards documents. This word is only meaningful on the first descriptor of a packet, it is ignored on subsequent descriptors.

*AAL5\_control (31:16):*

The 16 bit control field to be stuffed in the last cell of this packet. This field is a don't care for non AAL5 packets.

*AAL5\_len (15:0):*

The length in bytes of the AAL5 data, to be stuffed in the last cell of this packet. This field must be all zeros for non-AAL5 packets.



The 4 byte ATM header, without HEC field. For the exact meanings of the ATM header fields see the relevant CCITT/ITU standards documents. This word is only meaningful on the first descriptor of a packet, it is ignored on subsequent descriptors.

*pti-eop (31:29):*

The PTI to be stuffed into the last (EOP) cell of a packet.

*clp-eop (28):*

The CLP to be stuffed into the last (EOP) cell of a packet.

*vpi (27:20):*

The VPI to be stuffed into all cells in this packet.

*vci (19:4):*

The VCI to be stuffed into all cells in this packet.

*pti-neop (3:1):*

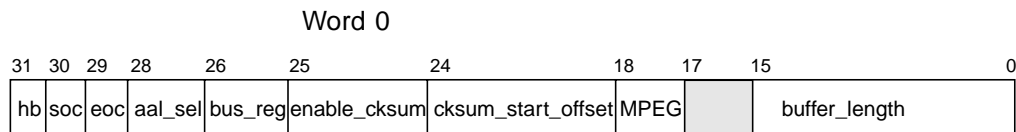
The PTI to be stuffed into all but the last cell of this packet.

*clp-neop (0):*

The CLP to be stuffed into all but the last cell of this packet.

### 2.2.1.2 Transmit Completion Ring

There is one completion ring shared by all transmit queues. The transmit completion ring is a direct ring.



*own (31):*

Set to 1 by the driver when the descriptor is handed to the hardware, cleared to 0 when the hardware posts.

The software must post descriptors to the completion ring by writing the descriptor offset number to the Completion Ring Kick Register and setting the own bit. The ATM622-s will check the Completion Ring Kick Register to make sure that it may write an entry and will clear the OWN bit.

*soc (30):*

The Start of Chain bit indicates that this is the first buffer in a chain of buffers to be gathered and sent as a single packet. Single buffer packets will have this bit set.

*eoc (29):*

The End of Chain bit indicates that this is the last buffer in a chain of buffers to be gathered and sent as a single packet. Single buffer packets will have this bit set.

*aal\_sel (28:27):*

This field selects which AAL framing should be used for this packet. The encodings are:

00 AAL5

01 Reserved

10 NULL/Software AAL

11 Reserved

The hardware will do all necessary padding and CRC generation for AAL5 packets. Hardware does nothing for Null AAL packets, allowing software to handle all necessary fields. Software must make sure that the packet is an exact multiple of 48 bytes. The reserved encoding is unused at this time. These bits are only meaningful on the first descriptor of a packet, it is ignored on subsequent descriptors.

*bus\_reg (26):*

The ATM622-s contains 2 registers of allowed bus transfer sizes, to represent two different devices in the system (see dynamic registers section). If set, SBus size register 1 will be used, if not set register 2 will be used. Each descriptor in a chain is allowed to use a different bus\_reg setting, for example to DMA the TCP & IP headers from the kernel memory and the packet data from an SBus framegrabber card.

*enable\_cksum (25):*

If set, the ATM622-s will stuff a TCP/UDP checksum calculated starting at cksum\_start\_offset and stuff it at cksum\_stuff\_offset. This bit is only meaningful on the first descriptor of a packet, it is ignored on subsequent descriptors. When this bit is active the entire packet must be written into External Memory before the checksum is completed and since CBR channels are generally allocated less than a packet's size worth of External Buffer Memory this bit should probably not be set for CBR channels.

*cksum\_start\_offset (24:19):*

This field should have all zeros.

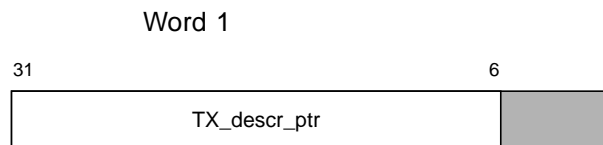
*MPEG (18):*

If set, this descriptor should be treated as an MPEG segmentation packet. The buffer length in word 0 is the length of a single MPEG frame, in bytes. The checksum stuff offset field in word 2 is redefined to be the number of MPEG frames in this descriptor. If the MPEG bit is set, the enable\_cksum bit in word 0 must be 0, you cannot use TCP checksumming on MPEG frames. The aal\_sel field must be set to 00, meaning AAL5, if the MPEG bit is set.

*reserved (17:16):*

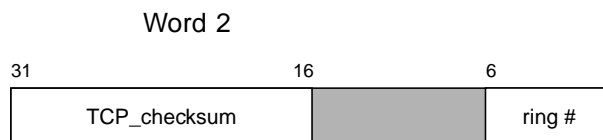
*buffer\_length (15:0):*

This field should have all zeros.



*TX\_descr\_ptr (31:6):*

Pointer to the last TX descriptor for this packet. The lower 6 bits of this word are not guaranteed to be zero and should be masked off by software.



*TCP\_checksum (31:16):*

The computed checksum which was stuffed into the outgoing packet. The TCP checksum is only valid for VBR channels, it is garbage for CBR and does not imply that a checksum was stuffed.

---

**Note:** This field is intended to aid in debugging the initial zero copy transmit software. It is not useful in normal chip operation.

---



*Ring Number (6:0):*

The ring number is a reference used to determine from which of the 127 data descriptor rings did this completion entry originate. This number along with a portion of the tx description pointer can be used as an index to identify the originating tx data descriptor. The portion of the tx descriptor used for identifying the originating tx data descriptor is equal to the number of bits (starting from bit 6) needed to encode the number of entries on the ring. For example if the ring number is 3 and the ring size is 64 then to determine the index look at bits 11 through 6 of the tx descriptor. The index is then ring number concatenated with bits 11 through 6.

*2.2.1.3 TX data buffers*

Data to be transmitted is held in buffers. There is no enforced structure to the buffers, they are treated as streams of bytes. The buffers can be in the kernel or user space, or even on another memory-mapped device in the system, such as another SBus card. The ATM622-s contains 2 separate registers of allowed bus transfer sizes and a bit in every descriptor to select between them to represent two distinct devices in the system. There are no alignment restrictions to buffer addresses, though for best performance it is recommended that buffers be 64 byte aligned.

*2.2.1.4 Receive Free Buffer Descriptor Ring*

Targeted VCs each have their own buffer descriptor ring. Non targeted VCs share a common kernel buffer ring. If a packet arrives for a targeted VC, and no buffer is available on the targeted buffer ring, the packet will be placed in a buffer from the kernel ring instead. Targeted and non-targeted buffer rings have the same structure, the following description applies to both types. The non-targeted ring has 1024 entries, each of the targeted rings has 64 entries. The free buffer ring is an indirect ring.

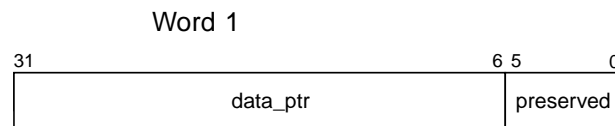


*own (31):*

Set to 1 by the driver to relinquish ownership to the hardware. Will be cleared by the hardware when it takes the buffer from the ring.

*buf\_len (9:0):*

Length of the data buffer, in units of 64 bytes. Receive data buffers must be a multiple of 64 bytes in size.

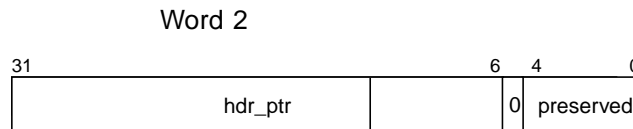


*data\_ptr (31:6):*

pointer to the data buffer. Receive data buffers must be 64 byte aligned.

*preserved (5:0):*

The hardware does not use these bits, and will set the lower 6 address bits to 0 when doing Sbus transfers. However the hardware will preserve whatever value is here all the way to the completion ring. It is recommended the driver use this value as an index into a table of buffers in use.



*hdr\_ptr (31:6):*

pointer to the header buffer. Receive header buffers must be 64 byte aligned. If the packet is small, up to 108 bytes of packet data may be placed in the header buffer in addition to the packet header. The header buffer is assumed to be large enough to hold the header, software must make sure the buffer is larger than the largest hlen programmed for any channel + 192 bytes (because the ATM622-s does 64 byte writes, and may do up to 3 writes of packet data to the header buffer).

*zero(5):*

Software must set this bit to zero.

*preserved (4:0):*

The hardware does not use these bits, and will set the lower 6 address bits to 0 when doing Sbus transfers. However the hardware will preserve whatever value is here all the way to the completion ring. It is recommended the driver use this value as an index into a table of buffers in use.

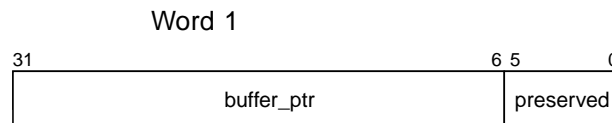
### 2.2.1.5 Targeted buffer completion information

When the ATM622-s fills a buffer picked up from a targeted buffer ring, it will write buffer status information back to the targeted ring. This allows buffer chaining on the targeted rings.



*own (31):*

Hardware will clear the own bit when writing to the descriptor.

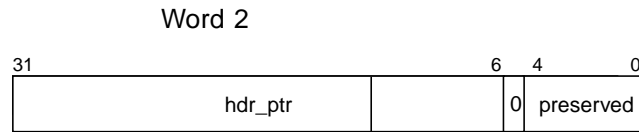


*buffer\_pointer (31:6):*

Pointer to the LAST byte of the packet. Software must determine the length of the packet based on the final\_buf\_length and subtract it from this pointer to get the first byte of the packet, or allocate all buffers with a particular alignment and mask off the lower order bits from this pointer.

*preserved (5:0):*

The ATM622-s preserves whatever bits were here from the RX buffer ring.



*header\_pointer (31:6):*

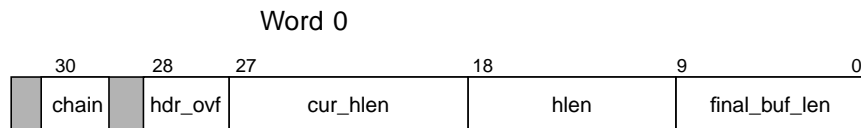
Pointer to the end of the packet header. The driver must subtract the length of the header from this pointer to back up to the beginning of the header. This field points to the header buffer which was picked up from the RX buffer ring for this data buffer. The header will be placed in the header buffer for the first descriptor in a chain; all subsequent descriptors point to unused header buffers which can be immediately reused.

*preserved(4:0):*

The ATM622-s preserves whatever bits were here from the RX buffer ring.

## 2.2.1.6 RX Completion Ring

The RX completion ring consists of 1024 entries each 64 bytes in size. The RX completion ring is a direct ring.



*chain (30):*

If set, indicates that this is the last buffer in a chain. If 0, indicates this is the first or subsequent buffer in a chain.

*hdr\_ovf (28):*

Indicates that an EOP was detected early in the packet, and so all packet data was placed in the header buffer. If this bit is set then cur\_hlen will contain the 1s complement of the number of words overflowed; i.e.  $\sim\text{cur\_hlen} + \text{hlen} = \text{length of packet}$ .

*cur\_hlen (27:19):*

If *hdr\_ovf* is not set, this is the number of words remaining to be split off in *hlen* when the packet ended. A zero indicates the header was split off exactly. If *hdr\_ovf* is set, this is the number of data words overflowed into the header buffer.

*hlen (18:10):*

Programmed by software when the VCI is set up. The number of 32 bit words to split off into the kernel header buffer for each packet.

*final\_buf\_length (9:0)*

Contains the amount of space left in the buffer when the packet ended, in units of 64 bytes. Software must keep track of the initial buffer length and subtract this length from it to get the received packet length.

Word 1

31	30	29	28	27	26	25	24	0
aal_sel	targeted	kernel_abort	pkt_crash	off	crc_err			

*aal\_sel (31:30):*

The AAL type programmed by software for this VCI.

*targeted (29):*

Indicates if software programmed this VCI to be targeted.

*kernel\_abort (28):*

If set, indicates this VCI was marked as targeted but no buffer was available on the targeted buffer ring, so a non-targeted buffer was used instead. If this bit is set on a buffer in the middle of a chain of buffers, all further buffers in this packet will also abort to the non-targeted buffer ring.

*pkt\_crash (27):*

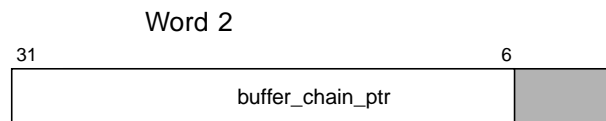
Indicates that the packet overflowed its buffer before the EOP was detected. This packet is corrupted, and the CRC error bit will also be set. If a *pkt\_crash* occurred software must not trust anything in words 5-7 of the completion ring (the ATM header, TCP checksum, and AAL5 information), as they are likely random garbage.

*off (26):*

If set, software turned this VCI off. The ATM622-s automatically returns any buffers when a VCI is turned off.

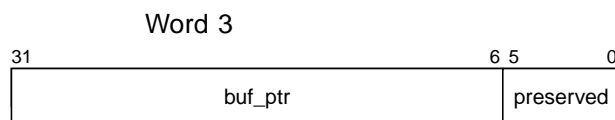
*crc\_err (25):*

If set, the CRC was incorrect on this packet. If this bit is set, the num\_cells field in word 6 is the number of cells the hardware actually received, not the correct number of cells the remote ATM host transmitted.



*buffer\_chain\_ptr (31:6):*

A 64 byte aligned pointer to the targeted buffer ring, points to the last descriptor in this packet. Software must back up the targeted buffer ring checking the chain bits til it finds the first buffer in the chain. This word is only valid for targeted VCIs, it is not valid for non-targeted VCIs.

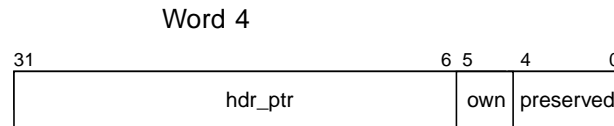


*buffer\_pointer (31:6):*

Pointer to the start of the last DMA burst for this packet. To find the beginning of the buffer it is recommended software use the lower 6 bits of this and the next word as an index into a table of allocated buffers.

*preserved (5:0):*

The ATM622-s preserves whatever bits were here from the RX buffer ring descriptor.

*header\_pointer (31:6):*

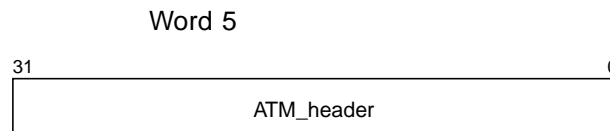
Pointer to the end of the last DMA burst for the header. This will only be used in the first descriptor of a chain of buffers. Subsequent header buffers are unused by the hardware.

*own (5):*

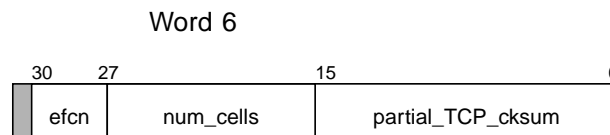
Cleared by the hardware whenever it writes to the completion ring.

*preserved (4:0):*

The ATM622-s preserves whatever bits were here from the RX buffer ring descriptor.

*ATM\_header (31:0):*

The 4 byte ATM header (sans HEC field) from the last cell of the received packet.

*efcn (30:27):*

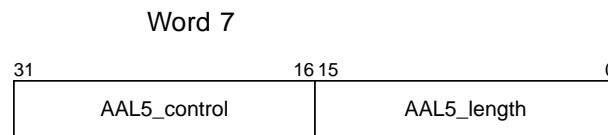
Number of cells from this packet which arrived with the EFCN bit set. If this value is 15 the counter overflowed, indicating at least 15 (and possibly more) cells arrived with the EFCN bit set. This field is invalid if there was a packet crash.

*num\_cells (26:16):*

The number of cells in this packet. This is only valid if there was no CRC error.

*TCP\_checksum (15:0):*

The TCP checksum calculated over the entire packet, including LLC/IP headers and AAL5 padding. Software must subtract out anything which was not supposed to be covered by the TCP checksum.



*AAL5 control (31:16):*

The 16 bit control field from the received packet AAL5 trailer. For non-AAL5 VCs this word is garbage.

*AAL5 length (15:0):*

The AAL5 length from the received packet. Note that if cells were dropped this will not be correct; software must look at `crc_err` before trusting this field.

### 2.2.1.7 RX buffers

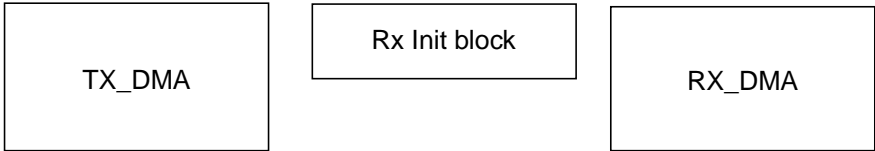
Received data, after the header is stripped off, is DMA'd into memory buffers. There is no enforced structure to the buffers, they are treated as streams of bytes. The buffers can be in the kernel or user space.

## 2.3 On-chip Control RAM Data Structure

The on-chip control RAM is organized into 3 separate areas: Rx Initialization Block, RX DMA state and Tx DMA state.



Figure 2-6 On chip control RAM regions

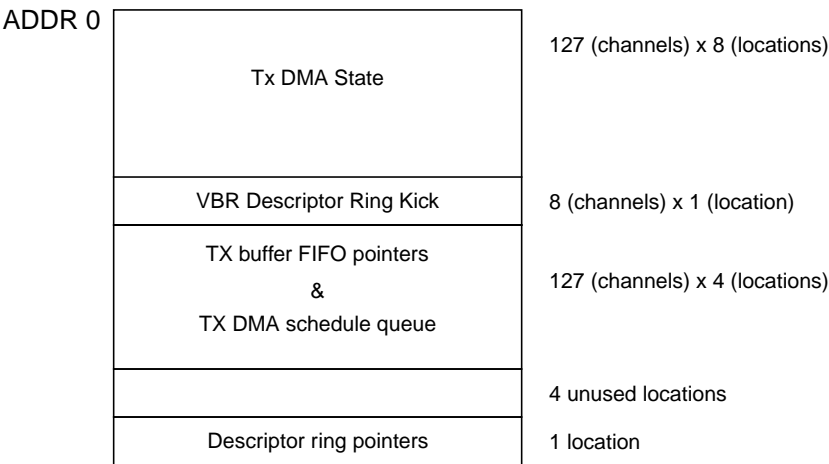


The RX DMA block contains state information specific to each of the 128 Receive VCI queues. The RX Init Block of the ATM622-s contains “global” state information. The TX\_DMA block contains state information specific to each of the 127 Transmit descriptor queues.

The TX\_DMA, RX\_Init and RX\_DMA RAMs are 32 bits wide.

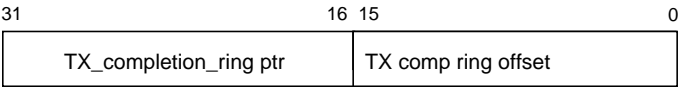
2.3.1 Transmit DMA Block

Figure 2-7 Tx DMA Control RAM



2.3.1.1 Descriptor ring pointers

*TX completion ring pointer*



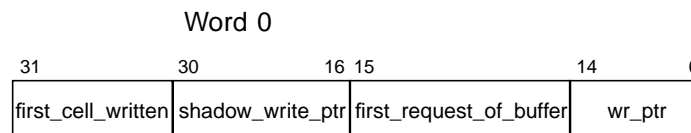
*TX\_completion\_ring\_ptr (31:16):*

pointer to the current location in the TX\_completion ring. This value is initialized by software at power on to the completion ring base. The completion ring is a fixed 1024 entries in size. The base of the ring must be 64 Kbyte aligned.

*TX\_completion\_ring\_offset(15:0):*

initially set to “0” by the software this field is incremented by the hardware after each write of a completion ring entry.

### 2.3.1.2 Transmit buffer FIFO pointers & Schedule Queue



*first\_cell\_written (31):*

Control bit used by the internal logic to indicate that the first complete cell of a packet has been written into the external tx buffer memory. This field is used by hardware only and should be initialized to zero.

*shadow\_wr\_ptr (30:16):*

The “actual” write pointer, pointing to the end of the currently DMA'd data. The wr\_ptr above will be updated from this pointer when the data is segmentable. This field is used by hardware only and should be initialized to zero.

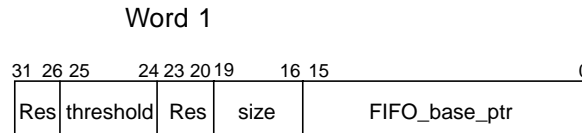
*first\_request\_of\_buffer (15):*

This bit indicates whether the first data request of a packet has been made by the tx load engine to the io block. It is used to so that the tx load engine will wait until after the 1st request is complete before making a second request. This bit must be initialized to a “1”.

*wr\_ptr (14:0):*

Address (in words) of the write pointer relative to the FIFO base. This pointer is word aligned. This is the write pointer the unload engine sees, and will at all times point to the end of segmentable data, while the shadow\_wr\_ptr below points to the actual end of the data. For example, a TCP packet cannot be seg-

mented until the entire packet has been DMA'd and the checksum calculated. This pointer will stay at the beginning of the TCP packet while the shadow\_wr\_ptr is incremented as data is DMA'd in, when the checksum is stuffed this pointer will be updated to point to the end of the packet. This field is used by hardware only and should be initialized to zero.



*reserved (31:26):*

Reserved by the hardware for future use. Should be set to 0 by software.

*threshold (25:24):*

If the channel is not scheduled and the amount of data in the external buffer is less than the amount of data defined by the fifo threshold field the unload block will schedule a DMA cycle.

**Table 2-1** FIFO Threshold

Threshold (25:24)	FIFO Threshold
00	undefined
01	3/4 full
10	1/2 full
11	1/4 full

*reserved (23:20):*

Reserved by the hardware for future use. Should be initialized to 0.

*size (19:16):*

Size of the FIFO (see following table for details). Initialized by software at power-on and not changed thereafter. It is recommended that VBR channels be allocated 16 Kbyte memory areas (to handle a 9180 byte TCP packet), and CBR channels be allocated 2 Kbyte memory areas.

**Table 2-2** Tx FIFO Size

Size (19:16)	FIFO Size
0	2K
1	4K
2	8K
3	16K
4	32K
5	64K
6	128K

*FIFO\_base\_ptr (15:0):*

Pointer (in words) to the beginning of the FIFO in the ATM622-s external memory space.

Word 2

31	21	20	19	18	17	16	15	14	2	1	0
cell count	AAL5	SOP	Load_Active	Unload_Active	Scheduled	extra_cell_needed	Read Pointer	reserved	MPEG		

*cell count (31:21):*

The Cell count is used internally by the unload engine during AAL5 packets to indicate the number of cells that will be transmitted. This field is used by hardware only and should be initialized to zero.

*AAL5 (20):*

The type of packet being read from External Memory is AAL5. This field is used by hardware only and should be initialized to zero.

*SOP (19):*

Start of Packet. When asserted this signal indicates that the 1st cell segmented from External Buffer is the Start of Packet. This field is used by hardware only and should be initialized to zero.

*Load\_Active (18):*

Load Engine Active. Indicates that there is data in host memory to send. This bit is set when the Host Writes to the Kick Register. It is cleared by the Load Engine when there is no more data in the Host to send. This field is used by hardware only and should be initialized to zero.

*Unload\_Active (17):*

Unload Active indicates that the Unload Engine is currently segmenting cells from the External Buffer or the channel needs to schedule a DMA cycle. It is cleared by the Unload Engine when the Load Engine is inactive and the External Buffer becomes Empty. In addition it is also cleared when a the Unload Engine has just scheduled a channel for DMA and the External Buffer is empty. This field is used by hardware only and should be initialized to zero.

*Scheduled (16):*

Scheduled. Indicates that the Unload Engine has already scheduled this channel for DMA, and should not schedule again. Will be set by the Unload Engine and cleared by the Load Engine. This field is used by hardware only and should be initialized to zero.

*extra\_cell\_needed (15):*

If extra cell needed is active the Unload Block will transmit another cell even though the external buffer for this channel may be empty. This field is used by hardware only and should be initialized to zero.

*rd\_ptr (14:0):*

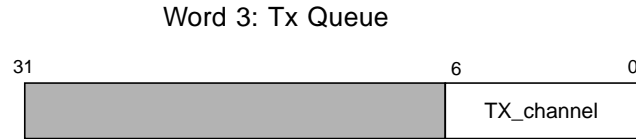
Address (in words) of the read pointer relative to the FIFO base. This pointer is word aligned. This field is used by hardware only and should be initialized to zero.

*reserved (1):*

Reserved for future use.

*MPEG (0):*

Used by the Tx Unload block to indicate segmentation for this channel is for a MPEG packet. This bit should be initialized to zero.



*reserved (31:7):*

Reserved by the hardware for future use.

*TX\_channel (6:0):*

Points to one of the 127 DMA channels, to be scheduled for DMA next. This field is used by hardware only and should be initialized to zero.

### 2.3.1.3 VBR Descriptor Ring Kick



*VBR Kick Value (12:0):*

Initially set to “0” by the software. This field is written to by hardware after every software write to the Tx Kick Register in which the descriptor ring value is from 0 to 7. These values represent the descriptor rings for the Variable Bit Rate Channels. Software does not need to modify this area of memory after initialization.

### 2.3.1.4 Tx DMA State

Summary of words:

Word 0: current buffer information

Word 1: current buffer pointer

Word 2: TCP information

Word 3: descriptor pointer

Word 4: ATM header

Word 5: AAL5 fields

Word 6: extra bytes register

Word 7: partial CRC

Word 0: Current Buffer Information

31	30	29	28	27	26	25	24	19	18	16	15	0
hb	soc	eoc	aal_sel	bus_reg	enable_cksum	cksum_start_offset					current_buffer_length	

*hb (31):*

Hardware bit. There is no own bit, but this bit should always be set to a “1”. This bit is used by the hardware internal to the ATM622-s. The hardware clears this bit **internally** only, to indicate that it is done with this descriptor entry.

*SOC (30):*

The Start of Chain bit indicates that this is the first buffer in a chain of buffers to be gathered and sent as a single packet. Single buffer packets will have this bit set.

*EOC (29):*

The End of Chain bit indicates that this is the last buffer in a chain of buffers to be gathered and sent as a single packet. Single buffer packets will have this bit set.

*aal\_sel (28:27):*

This field selects which AAL framing should be used for this packet. The encodings are:

00 AAL5

01 Reserved

10 NULL/Software AAL

11 Reserved

The hardware will do all necessary padding and CRC generation for AAL5 packets. Hardware does nothing for Null AAL packets, allowing software to handle all necessary fields. The reserved encoding is unused at this time. This setting is only meaningful on the first descriptor of a packet, it is ignored on subsequent descriptors.

*bus\_reg (26):*

The ATM622-s contains 2 registers of allowed bus transfer sizes, to represent two different devices in the system (one being host memory, the other being an alternate data source like a digitizer). This bit indicates which register to use for this buffer. Each buffer in a packet chain is allowed to use a different bus\_reg setting.

*enable\_cksum (25):*

If set the computed TCP checksum will be stuffed into the packet before transmission at an offset of cksum\_stuff\_offset.

*cksum\_start\_offset (24:19):*

Gives the number of half words from the beginning of the packet to skip before beginning the TCP/UDP checksum. Only valid if the enable\_cksum bit is set, it is ignored on subsequent descriptors.

*reserved (18:16):*

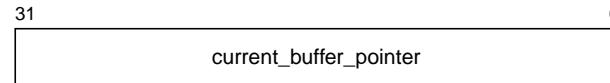
Reserved by the hardware for future use.

*buffer\_length (15:0):*

Specifies the number of bytes to be transmitted from this buffer. The maximum buffer size is 64 K bytes. This field is decremented by the hardware as cells are DMA'd into the local buffers.

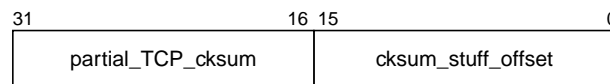


## Word 1: Current Buffer Pointer

*current\_buffer\_pointer (31:0):*

The current position in the transmit buffer. There are no alignment restrictions, the buffer can be byte aligned.

## Word 2: TCP Information

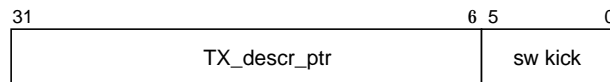
*partial\_TCP\_cksum (31:16):*

Calculated on the fly as the packet is DMA'd into the transmit buffer, the partial checksum for the current packet is stored here.

*cksum\_stuff\_offset (15:0):*

Where in the packet the completed TCP checksum should be stuffed. This field is only valid if the enable\_cksum bit is set in word 0.

## Word 3: Descriptor Pointer

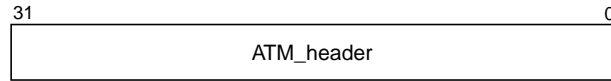
*TX\_descr\_ptr (31:6):*

Pointer to the descriptor for this BWG. Programmed by software at BWG activation. Must be 16 word aligned (each descriptor size is 64 bytes). When the tx\_descr\_ptr, bits <11:6>, match sw kick ptr it is implied that this is last descriptor posted by the sw. The sw kick must be greater than bits <11:6> to indicate that there are one or more descriptors have been queued by sw.

*sw kick (5:0):*

Software kick - records the number of the most recently written descriptor to the TX\_kick register for this descriptor. The ATM622-s will stop transmitting when it finishes the descriptor pointed to by kick. See section 2.1.3 for details. This is only valid for CBR channels.

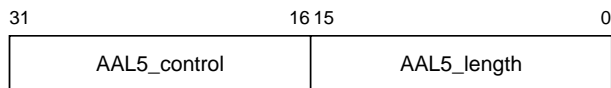
Word 4: ATM Header



*ATM\_header (31:0):*

Programmed in the TX descriptor, this will be written into the TX buffer by the DMA state machine as part of the header information, then written into this word by the segmentation state machine. See word 3 in the TX descriptor for a complete explanation of the format of this word, section 2.2.1.1.

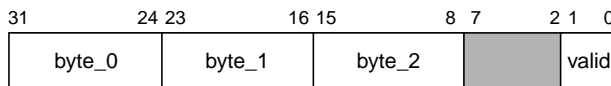
Word 5: AAL5 Fields



*AAL5\_control & AAL5\_length:*

Programmed in the TX descriptor, this will be written into the TX buffer by the DMA state machine, then written into this word by the segmentation state machine.

Word 6: Extra Bytes Register



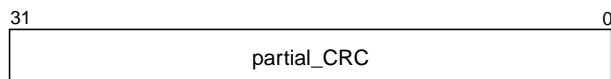
*byte\_0(31:24), byte\_1 (23:16), byte\_2(15:8):*

If a transmit DMA read ends at an odd byte boundary in the external buffer memory, the odd bytes will be stored here until the next DMA read to fill the word.

*valid (1:0):*

Indicates how many of the three bytes of data are valid.

Word 7: Partial CRC



*partial\_AAL5\_CRC* (31:0):

The CRC is calculated on the fly and stuffed into the last cell of the packet. The partial CRC is stored here.

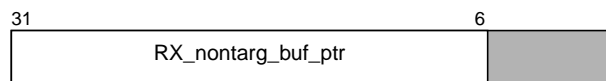
### 2.3.2 Rx Initialization Block

**Figure 2-8** Logical Init block regions

Descriptor ring pointers	3
RX DMA schedule queue	32
RX VCI-DMA state map	256
RX LRU list	128
RX bucket next pointers	1024

This is only a logical layout. In actually physical layout these regions are split between two blocks of RAM. See the I/O address map for details.

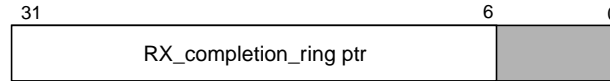
#### 2.3.2.1 Descriptor ring pointers



*RX\_nontarg\_buf\_ptr* (31:6):

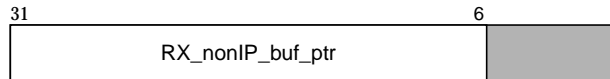
Pointer to the current location in the RX non-targeted data buffer ring. This value is initialized by software at poweron to the buffer ring base, and is incremented during chip operation. The buffer ring is a fixed 1024 entries in size, and must be 64 Kbyte aligned, the lower 6 bits are masked to 0 by the hardware.

*RX completion ring pointer*



*RX\_completion\_ring\_ptr (31:6):*

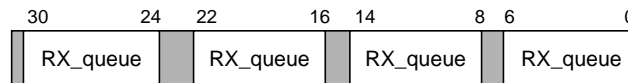
Pointer to the current location in the RX completion ring. This value is initialized by software at power on to the completion ring base, and is incremented during chip operation. The completion ring is a fixed 1024 entries in size. The base of the ring must be 64 Kbyte aligned, the lower six bits are masked to 0 by the hardware.



*RX\_nonIP\_buf\_ptr (31:6):*

Pointer to the current location in the RX nonIP data buffer ring. This value is initialized by software at poweron to the buffer ring base, and is incremented during chip operation. The buffer ring is a fixed 1024 entries in size, and must be 64 Kbyte aligned, the lower 6 bits are masked to 0 by the hardware.

### 2.3.2.2 RX DMA schedule queue entry



*RX\_queue (30:24):*

*RX\_queue (22:16):*

*RX\_queue (14:8):*

*RX\_queue (6:0):*

Which of the 128 RX queues should be scheduled for DMA next. The schedule entries are packed in 4 per 32 bit word to save space.

*Schedule queue Initialization:*

No initialization is necessary in the schedule queue.

**2.3.2.3 RX VCI-DMA State map**

31	30	24	23	22	16	15	14	8	7	6	0
cached	DMA_state	cached	DMA_state	cached	DMA_state	cached	DMA_state	cached	DMA_state		

There is one entry in the DMA state map for each of the 1024 active VCIs. An incoming cell's VCI will be looked up in this table to determine if the DMA state for this VCI is on chip and if it is, where in the state memory it resides. The VCI's are packed in 4 per 32 bit word in the control memory word to conserve space. Thus to look up VCI 7, you read word 1 and look in byte 3 (VCIs start at 0).

On poweron the software must initialize all bits to 0.

*cached (31):*

If set, the VCI's state information is currently cached on chip, and the DMA\_state field is valid. If not set, the VCI's state is off chip in the external RAM.

*DMA\_state (30:24):*

If the cached bit is set, this field points to which of the 128 DMA states this VCI should map into. If the cached bit is not set, then the lower 7 bits contain information about the VCI: This byte is written by software to turn a channel on and adjusted by hardware thereafter.

Non-cached map entry:

7		2	0
0		aal_sel	0 on

*VCI map Initialization:*

Software must initialize all entries except VCI #0 to 0x00. VCI #0 should be initialized 0x80, meaning cached in DMA state 0.

### 2.3.2.4 RX LRU linked list



The ATM622-s can have 1024 active receive VCIs, of which the DMA state information for 128 are cached on chip, with the remainder held in the external SS-RAM. DMA state caches are assigned and reclaimed in strict Least Recently Used fashion. This is done by maintaining a doubly linked list of all DMA states, and moving the entry for a given DMA state to the head of the list when a cell arrives for that DMA state. The entry at the end of the list is the least recently used entry, and the first candidate for reclamation.

#### *used (31):*

Indicates that the DMA state is currently occupied and the VCI field is valid. This bit is used in the flush and update mechanism to dump an old DMA state to external memory and read in a new one. Hardware must update this field if the VCI is turned on or off.

#### *VCI (25:16):*

Indicates which VCI is currently in the DMA state. Used when a DMA state must be flushed, to determine where in the external memory the data should be stored.

#### *prev (14:8):*

Pointer to the previous DMA state in the linked list.

#### *next (6:0):*

Pointer to the next DMA state in the linked list.

#### *LRU list initialization:*

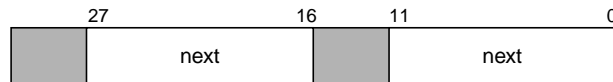
DMA state number zero is used for OAM cells. It is never flushed out and is not linked on the LRU list. DMA state number zero should be initialized to 0x80000000, meaning used, VCI number 0, prev = next = 0.

The other LRU entries are initialized as a doubly linked list. LRU entry #1 should be the first item on the list, with prev = 0 and next = 2. LRU entry #127 should be the last entry on the list, with prev = 126 and next = 0. All other entries i should have prev = i-1 and next = i+1.

Software can 'lock' other DMA states into the cache, so they will never be flushed out, by setting the prev and next pointers to 0. Software must start from the highest DMA states when locking things into the cache. For example, if it is desired to have two more VCIs locked on to the chip, DMA states #1 and #2 must be chosen (#0 is always locked on the chip, remember). In this case software must write the DMA state information directly into the on-chip cache (not the external RAM as is normally done), and must write to the map entry to place a 1 in the cached bit and the DMA state number into the map. It is recommended that VCIs used for F5 OAM information (3 & 4) be locked onto the chip.

### 2.3.2.5 RX buckets next pointers

Each of the 48 byte buckets has a pointer to the next bucket in the linked list (either the free list or one of the 128 RX channel lists). The next pointers are stored on chip in the init block because updating them requires a read and a write, which is expensive to the external RAM. The next pointers are packed in 2 per word to conserve memory. A next pointer equal to zero indicates end of list. See section 2.4.2.2 for details of how these pointers are used.



*next (27:16):*

Pointer to the next bucket.

*next (11:0):*

Pointer to the next bucket.

*Free list initialization:*

Bucket #0 is unused, and **must** have its next pointer initialized to zero. Bucket #1 must be initialized as the rf\_rl\_bucket, and must have its next pointer set to zero. Bucket number 2 is initialized as first\_free, and should have its next pointer set to 3. All other buckets *i* are on the free list, and have their next pointers = *i*+1 until bucket number 2047, which has a next pointer of zero (end of list).

Software must initialize the rf\_rl\_bucket and first\_free registers to 1 and 2, respectively. Software must also initialize the num\_free counter to 2047.

### 2.3.3 Receive DMA Block

This block contains the state information for 1024 receive virtual channels. Of these, 128 states are cached on the chip, and the remainder are kept in external memory. Each entry consists of 8 words.

Word 0: current packet reassembly information

Word 1: partial CRC

Word 2: current packet lengths and status

Word 3: bucket chain pointers

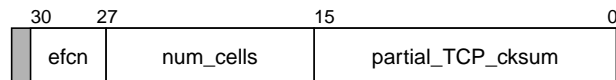
Word 4: partial bucket pointers

Word 5: targeted buffer ring ptr

Word 6: header buffer pointer

Word 7: data buffer pointer

Word 0: Current Packet Reassembly Information



*efcn (30:27):*

The number of cells in this packet which arrived with the EFCN bit set.

*num\_cells (26:16):*

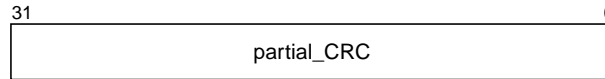
For NULL AAL connections, stores the number of cells remaining to be received for the current packet. This field is loaded by the hardware from word 4 at the end of each packet, and incremented with each arriving cell. Software should initialize this field with the same value as in word 4 num\_cells.

*partial\_TCP\_cksum (15:0):*

The partial TCP checksum up to the last byte of the most recently received cell.

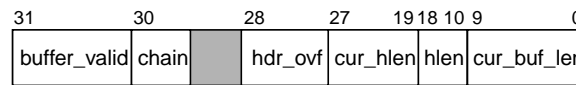


Word 1: Partial CRC

*partial\_AAL5\_CRC (31:0):*

Storage area for the AAL5 CRC while it is computed. At the End of Packet, this will be used to determine if the packet arrived without error.

Word 2: Current Packet Lengths and Status

*buffer\_valid (31):*

The buffer address and length are valid. This bit is read as the own bit from the RX buffer descriptor. It will be cleared when the buffer is returned to the RX completion ring.

*chain (30):*

Will be set when the ATM622-s detects EOP. This word is written to the RX completion ring, the chain bit is used to demarcate chains of buffers for a single packet.

*hdr\_ovf (28):*

Indicates that the entire header was split off into the header buffer. Depending on the length of the packet, a few words of data might also have been placed in the header buffer, which can be determined from the cur\_hlen field. The way to use these fields is:

hdr_ovf	cur_hlen	Comments
hdr_ovf=0	cur_hlen = 0	Software indicated no header was to be split off.
hdr_ovf=0	cur_hlen!= 0	Packet ended before header was split off. Look at aal5_len or num_cells to find the actual packet length.
hdr_ovf=1	cur_hlen = 0	The header was split off exactly, the rest of the packet (if any) is in the data buffer.
hdr_ovf=1	cur_hlen!= 0	Packet ended shortly after the header, and hardware aborted all packet data into the header buffer. Look at aal5 len or num_cells for the actual packet length.

*cur\_hlen (27:19):*

When the first cell of a packet arrives, the `hdr_len` field is loaded into this field, and decremented as words are DMA'd into the header buffer.

*hlen (18:10):*

Programmed by software when the VCI is set up. The number of 32 bit words to split off into the kernel header buffer for each packet.

*current\_buffer\_length (9:0):*

Count of the length of the current receive buffer in units of 64 bytes. Decrement-ed by the hardware as bytes are DMA'd into memory.

Word 3: Bucket Chain Pointers

31	30	29	28	27	26	25	24	23	12	11	0
aal_sel	targeted	kernel_abort	pkt_crash	off	dirty	S	last_bucket			first_bucket	

*aal\_sel (31:30):*

Encoding of AAL type, used for EOP detection. Programmed as follows:

00 AAL5

01 Reserved

10 NULL AAL

11 AAL 3/4

For AAL5, EOP detection is based on the encoding of the PTI bits as defined in the CCITT standards. For the NULL AAL, EOP detection is based on the number of bytes (when the buffer fills, it is handed back to the software via the completion ring). For AAL 3/4, EOP is detected based on the encoding of bit 6 as defined in the CCITT documents.

*targeted (29):*

If set, this VCI is targeted and the `buffer_chain_ptr` is valid. If not set and the `kernel_abort` bit is set, the non-IP ring will be used. If `targeted` is 0 and `kernel_abort` is 0, the non-targeted buffer ring will be used.

*kernel\_abort (28):*

Indicates that the VCI was marked as targeted, but no buffer was available on the targeted buffer ring when the first cell of this packet arrived so the ATM622-s aborted to a kernel buffer ring.

*pkt\_crash (27):*

Will be set if no buffer is available for the current packet, causing the rest of the packet to be discarded.

*off (26):*

Indicates the VCI has been turned off.

*dirty (25):*

Indicates the cached DMA state has been modified since it was read in from external RAM. If this DMA state is dropped from the cache it must be written to external RAM first.

*S (24):*

Indicates that this receive channel has been scheduled for DMA. The reassembly state machine will not schedule this channel for DMA again if the S bit is set. To avoid race conditions, this bit must be set via an atomic read-modify-write cycle.

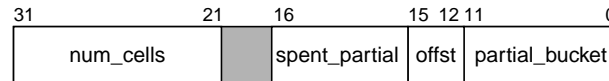
*last\_bucket (23:12):*

Pointer to the last bucket in the linked list for this VCI. If this pointer is zero, the list is empty. If this field is nonzero and not equal to the first bucket, then at least 3 cells (144 bytes) are present and the channel will be scheduled for DMA by the RX reassembly state machine.

*first\_bucket (11:0):*

Pointer to the first bucket in the linked list for this VCI. If this pointer is zero, the list is empty.

Word 4: Partial Bucket Pointers



*num\_cells (31:21):*

For a NULL AAL connection, gives the number of cells to receive before signaling EOP and writing to the completion ring. This field is programmed in 2's complement. Software must program this field at VCI setup, it cannot be adjusted thereafter. For AAL34 or AAL5 connections this field should be 0.

*spentpartial (16):*

Indicates there is a partial bucket which has been completely DMA'd but not yet returned to the free list.

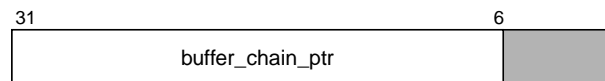
*offset (15:12):*

Number of bytes read in the partial bucket.

*partial\_bucket (11:0):*

Pointer to a partially read bucket. When the RX system side DMA's data to memory, it will not necessarily DMA in cell-sized chunks. If at the end of a DMA burst a cell is partially read, it will be removed from the linked list and placed here. The number of bytes which have already been DMA'd from this bucket is stored in the offset field.

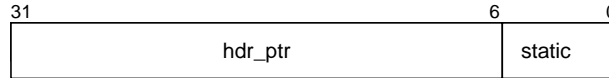
Word 5: Targeted Buffer Ring Pointer



*buffer\_chain\_pointer (31:6):*

For a targeted buffer, this points to the current descriptor in the buffer ring for this VCI. This word is unused for non-targeted VCs. This word is written to the RX completion ring at the end of a packet. This pointer is 64 byte aligned.

Word 6: Header Buffer Pointer



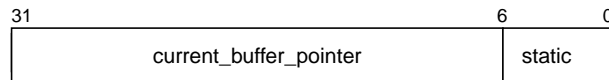
*header\_pointer (31:6):*

Pointer to the buffer for the header.

*static (5:0):*

The lower 6 bits of this word will be picked up from the RX buffer ring, stored on chip during DMA, and returned unmodified to the RX completion ring. Software can use these bits for any purpose.

Word 7: Data Buffer Pointer



*current\_buffer\_pointer (31:6):*

Pointer to the data buffer.

*static (5:0):*

The lower 6 bits of this word will be picked up from the RX buffer ring, stored on chip during DMA, and returned unmodified to the RX completion ring. Software can use these bits for any purpose.

## 2.4 Off Chip memory

The ATM622-s requires external SRAM for operation. The SRAM is divided into two addressable ports, one for transmit and one for receive. Software can access the external memory for diagnostic purposes (single word accesses only).

## 2.4.1 Transmit Memory

### 2.4.1.1 Transmit Bandwidth Group Table

Packets are transmitted based on the Bandwidth Group Table and the Leaky Bucket Registers. The Unload Engine first looks into the Bandwidth Group Table to determine which channel to segment. If the entry in the table is "0" the Unload Engine will transmit the highest priority Leaky Bucket channel. Finally if there are not any Leaky Bucket channels requesting to transmit the Unload Engine will wait a Link Rate time before looking at the next entry in the Bandwidth Group Table. Since an entry is visited by the Unload Engine at the programmed Link Rate the Tx Bandwidth Group Table provides an excellent means to provide Constant Bit Rate service.

The Bandwidth Group Table begins at address "0" of the Tx External Buffer and can be programmed to a maximum length of 4096 words. A Bandwidth Group Table word contain 4 entries, at the maximum size the Bandwidth Group Table has 16384 entries.

Tx Bandwidth Group Table Word



*reserved (31,23,15,7):*

Reserved by the hardware for future use.

*entry0 (30:24):*

Entry0 is the first entry of the Tx Bandwidth Group Table Word to be segmented.

*entry1 (22:16):*

Entry1 is the second entry of the Tx Bandwidth Group Table Word to be segmented.

*entry2 (14:8):*

Entry2 is the third entry of the Tx Bandwidth Group Table Word to be segmented.

*entry3 (6:0):*

Entry3 is the last entry of the Tx Bandwidth Group Table Word to be segmented.

### 2.4.1.2 Transmit buffer memory

Packet data is moved from Host Memory to the Transmit Buffer Memory. There are three types of packets that may be moved from Host Memory. The type of packets are AAL5 without MPEG Packetization, AAL5 with MPEG Packetization and non-AAL5. The type of packet is determined by the data descriptor. As soon as data is moved into the transmit buffer memory it may be moved out by the Tx Unload Block to be segmented into cells, except for TCP packets which are fully buffered in the Transmit buffer memory before transmission. Each transmit descriptor has its own area of buffer memory, up to 128Kbytes in size. Multiple packets from that descriptor can be held in the buffer at any given time. Each packet in the buffer memory is preceded by a header containing the information the segmentation state machine will need in order to form ATM cells.

For non-MPEG AAL5 packets the header tag bits are set to "1" for the first two words. The first word contains AAL5 control and length fields. The second word contains the ATM Header (See Figure 2-9). The header is then followed by two empty words and the data section begins. The data ends when either of the tag bits is set to a "1".

In the case of MPEG AAL5 packets the header tag bits are set to "0" for the first word and a "1" for the second word. The rest of the header is the same as the non-MPEG AAL5 packet. The data also begins in fifth word from the beginning like the non-MPEG packet. A packet ends with one tag bit set to "1" and one tag bit set to "0". But unlike the non-MPEG packet a new header does not follow. The Tx Unload Block will use the previous header to create cells for this packet. The previous header will be reused until both tag bits are set to a "1". When both tag bits are set to a "1" a new header will follow.

For non-AAL5 packets the header tag bits of the first word are a don't care and the second word's tag bits are set to a "0". The first word is not used but the length field must be zero. The second word is the ATM Header. Data follows on the fifth word and ends when either tag bit is set to a "1". Actual data size must be a multiple of 48 bytes.

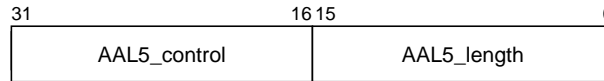
**Figure 2-9** TX buffer memory format

AAL5 - Non MPEG				AAL5 - MPEG				Non-AAL5			
tags				tags				tags			
1	1	control	length	0	0	control	length	x	x	-----	0000
1	1	ATM_header		1	1	ATM_header		0	0	ATM_header	
x	x	-----		x	x	-----		x	x	-----	
x	x	-----		x	x	-----		x	x	-----	
0	0	packet data 1		0	0	packet data 1 of descriptor 1		0	0	packet data 1	
0	0			0	0			0	0		
0	0			0	0			0	0		
0	0			0	0			0	0		
1	1	Last byte(s) of data		1	0	Last byte(s) of data		1	1	Last byte(s) of data	
x	x	-----		x	x	-----		x	x	-----	
x	x	-----		x	x	-----		x	x	-----	
x	x	-----		x	x	-----		x	x	-----	
1	1	control	length	0	0	packet data 2 of descriptor 1		x	x	-----	0000
1	1	ATM_header		0	0			0	0	ATM_header	
x	x	-----		0	0			x	x	-----	
x	x	-----		0	0			x	x	-----	
0	0	packet data 2		1	1	Last byte(s) of data		0	0	packet data 2	
0	0			0	0	control    length		0	0		
0	0			1	1	ATM_header		0	0		
0	0			0	0	-----		0	0		
1	1	Last byte(s) of data		1	1	-----		1	1	Last byte(s) of data	
				0	0	packet data 1 of descriptor 2					
				0	0						
				0	0						
				0	0						
				1	1	Last byte(s) of data					

The fields in this buffer header are entirely under the control of the hardware and should not be modified by software. This information is presented for reference and debugging purposes only.

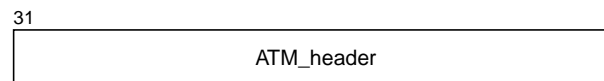


Word 0: AAL5 fields



If present, this word indicates that the packet is an AAL5 packet, and that the control, length and CRC fields should be stuffed into the last cell on transmission. The hardware can tell if the word is present or absent by looking at the TAG bit. A 1 indicates that the AAL5 word is present, 0 indicates that this is a non-AAL5 packet and user data starts at the next location. This word is carried directly from word 4 of the transmit descriptor (section 2.2.1.1).

Word 1: ATM header



#### *ATM cell header (31:0):*

This word contains the ATM cell header to use for the transmitted cells, along with the PTI encoding for the first and last cells. These bits are carried directly from the transmit descriptor word 3, see that section for the detailed encoding of the header field.

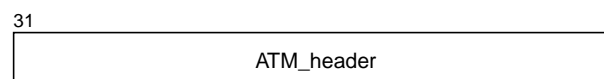
## 2.4.2 Receive memory

### 2.4.2.1 Rx bucket memory

There is a certain amount of status information which needs to be propagated to the system software for every received packet, including checksums, header bytes, and congestion information. When an EOP cell is detected, the RX media stores this information into an additional bucket appended to the packet in the external RX memory. There are 4 words of status information for each received packet.

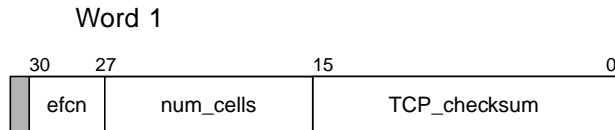
The trailer in the RX bucket is entirely under the control of the hardware, and is not intended to be modified by software. This information is presented for debugging purposes. This information is available in the RX completion ring.

Word 0



*ATM\_header (31:0):*

The 4 byte ATM header from the last cell of the packet.



*efcn (30:27):*

Number of cells from this packet which arrived with the EFCN bit set. If this value is 0x15 then at least 15 (and possibly more) cells arrived with the EFCN bit set. The counter does not overflow, it merely stops counting at 15.

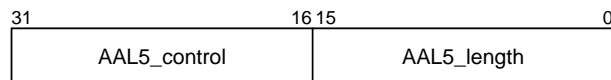
*num\_cells (26:16):*

The number of cells in this packet. Note that cells can be dropped in the case of packet crash after they have been counted here. Software must only use this value if there was no CRC error.

*TCP\_checksum (15:0):*

The TCP checksum calculated over the entire packet, including LLC/IP headers and AAL5 padding. Software must subtract out anything which was not supposed to be covered by the TCP checksum.

Word 2



*AAL 5 control (31:16)& AAL5 length (15:0):*

The AAL5 control and length fields are copied from the EOP cell and placed in the status bucket for eventual DMA to the completion ring. This is done for software efficiency, so that the length field will be easily available in the completion ring rather than buried at the end of the packet buffer. For non-AAL5 cells this is whatever happened to be in the next to last word of the last cell of the packet, and is unlikely to be useful.

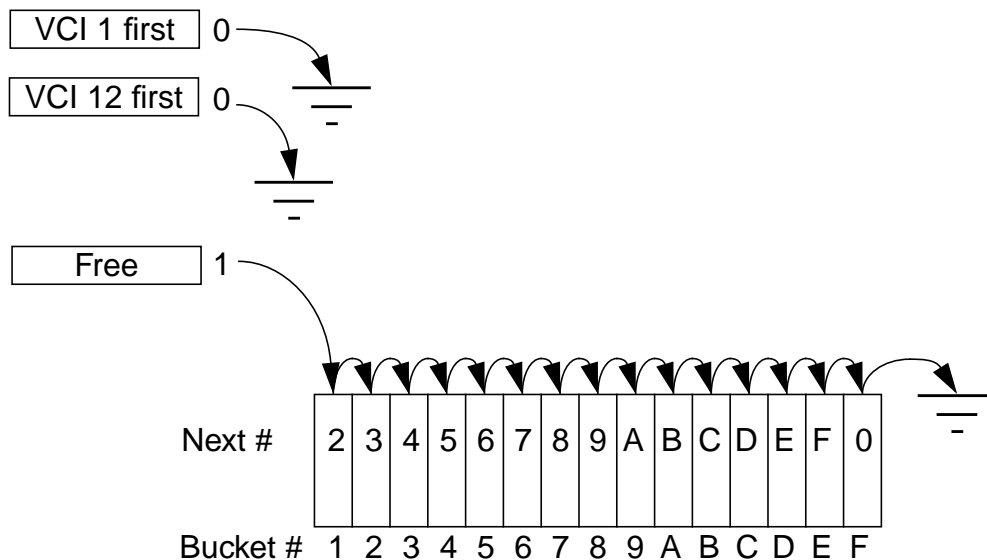
### 2.4.2.2 Receive buffer memory management

The external RX memory is managed as a large number of data “buckets”, where each bucket can hold the contents of a single ATM cell (i.e. 48 bytes). Buckets holding cells for a given VCI are placed on a linked list for eventual DMA to memory. Buckets which are not currently allocated to a VCI are kept on a free list.

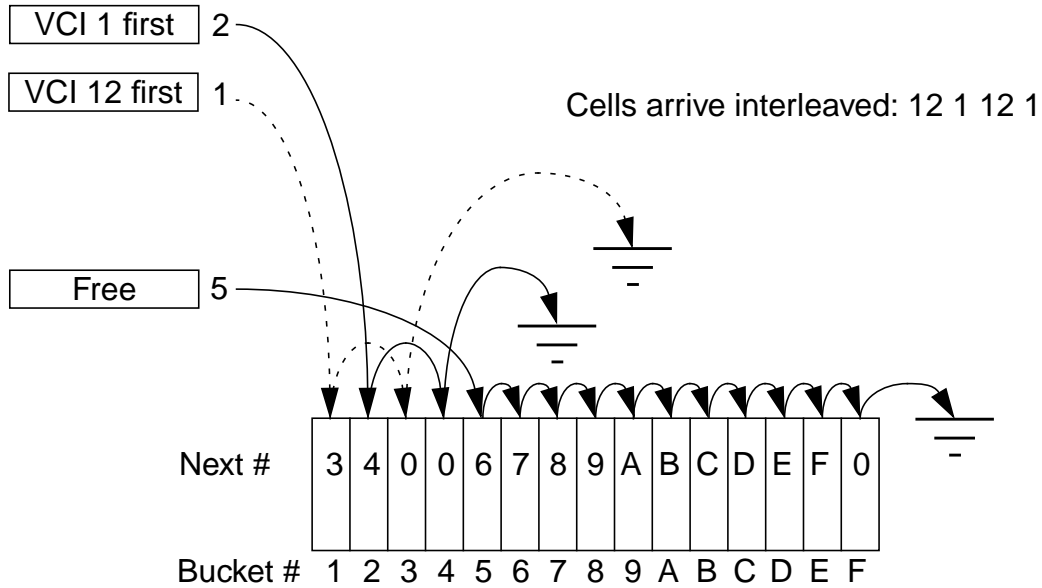
The pointers for the linked list are kept in a block of memory on chip. This is done for efficiency since the pointers are adjusted at every cell time. Every external bucket has a next pointer. Each VCI has a pointer to the first and last buckets in their linked list, and there is an additional pointer to the beginning of the free list. Bucket (and pointer) number 0 does not exist and is used to terminate a linked list (in software terms, a NULL pointer).

At power on software must initialize the next pointers for all buckets to place them on the linked list of free cells (figure 2-10). As cells arrive they are taken from the linked list and appended to the linked list for the VCI (figure 2-11). As data is DMA'd to host memory, cells are taken from the VCI list and placed at the beginning of the free list (figure 2-12).

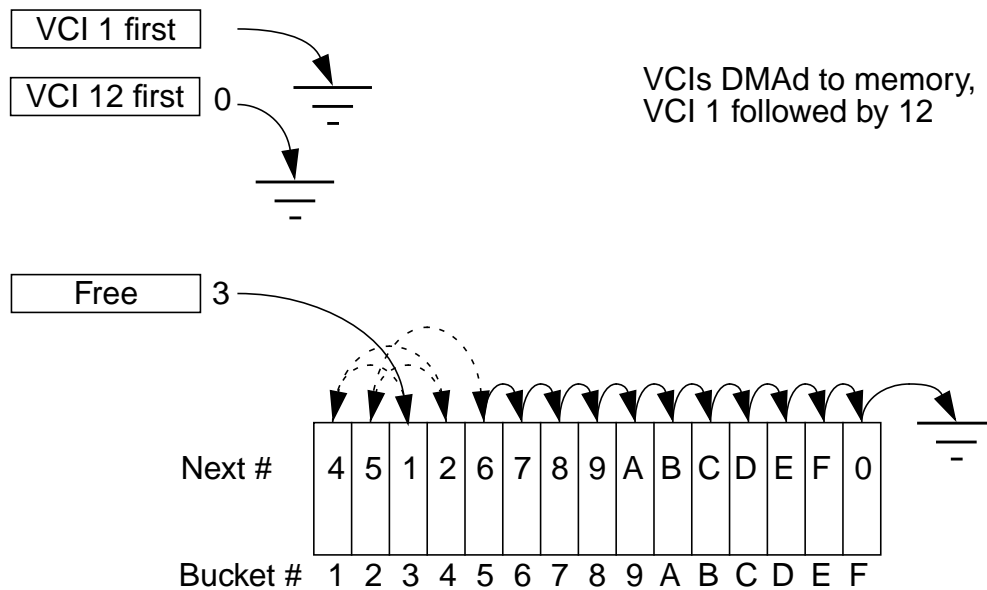
**Figure 2-10** All buckets on free list



**Figure 2-11** Cells arrive, buckets are taken from free list and placed on VCI lists.



**Figure 2-12** Data DMA'd to memory, buckets are placed at the head of the free list.



### 2.4.2.3 RX DMA state

Each of the ATM622-s 1024 receive VCIs has 8 words of state information, stored here. The 128 most active VCI's state information is cached on chip. Each VCI's entry is 8 words long, the format of the state information is described in section 2.3.3.

## 2.5 Registers, I/O Commands

### 2.5.1 Dynamic Registers

#### 2.5.1.1 Global registers & IO Commands

*Config*

9	8	7	6	5	4	3	2	1	0
	bm_tag_rd	rx_target_ring_size_64/128	155/622_mode	bm_tag_wr	rx_hcs_enable	tx_intr_mode	disable_loopback	en_tx	en_rx

The Configuration register (except bm\_tag\_rd) is cleared upon reset. All bits are write/read except bm\_tag\_rd bit (read only).

*bm\_tag\_rd (9:8):*

The two tag bits represent the tx (two tag bits) or rx buffer memory tag bit (only one tag bit, bm\_tag\_rd[8]) that are being read during Host slave read access. These bits are read only and are not changed by an SBus slave write, only by reading from external RAM.

*rx\_target\_ring\_size\_64/128 (7):*

This bit, when set, indicates that the size of free target buffer ring (all target rings) is 128 descriptors. It is otherwise 64 descriptors.

*155/622\_mode (6):*

When this bit is zero, the ATM622-s configures its utopia interface for 155 Mbit using octet data streams at mclk/2. When set, the 622 Mbit utopia interface is configured with 16 bit data stream at mclk speed. It is recommended that this bit is programmed while the chip is in loop back mode.

*bm\_tag\_wr (5):*

The *bm\_tag\_wr* represents the tag value that is written to tx or rx buffer memory. The software needs to access the buffer memory for some initialization, diagnostic and possibly for initial bring-up debugging. All subsequent writes to tx and rx buffer memory will write 1 or 0 to corresponding memory location tag bit depending on *bm\_tag\_wr* value. There will be two separate IO commands (*rx\_bm\_tag\_reg* and *tx\_bm\_tag\_reg*) to read the tag value at the memory location being read through SBus data bit 0 (*sb\_d[0]*).

---

**Note:** *bm\_tag\_wr* value gets written to both tag bits of TX buffer memory.

---

*rx\_hcs\_enable (4):*

When set, the ATM cell HCS checking is enabled on the receive path in the ATM622-s. Cells with bad HCS will be dropped and not pushed to the ATM622-s internal RX\_FIFO.

*tx\_intr\_mode (3):*

When set indicates that *tx\_individual* interrupt mode is selected. When clear indicates that *tx\_all* interrupt mode is selected. See the status register definition for more detail description.

*disable\_loopback (2):*

The ATM622-s contains an internal loopback for testing purposes. If this bit is set, the loopback will be disabled; it is otherwise enabled. In loop back mode, the ATM622-s transmit UTOPIA port will be disabled (no outgoing cells) and cells will be discarded at the receive's UTOPIA port. In loopback mode, the ATM622-s internal UTOPIA cell fifo (RX\_FIFO) will be linked to the ATM622-s internal transmit cell fifo (TX\_FIFO).

---

**Note:** There must be a hardware or software reset prior to enabling the internal loopback operation (*disable\_loopback* = 0).

---

*en\_rx (1):*

If set, enables operation of the receive side. All RX control memory fields must be properly initialized before this bit is set. This bit is cleared on reset.

*en\_tx (0):*

If set, enables operation of the transmit side. All TX control memory fields must be properly initialized before this bit is set. This bit is cleared on reset.

### *I/O Configuration Register*

9	8	7	6	5	4	3	2	1	0	
	par_en	skip_retry	64bit_1	16word_1	8word_1	4word_1	64bit_0	16word_0	8word_0	4word_0

The Config. register in the IO deals with IO specific details such as IO width, burst sizes, parity, and other IO specific parameters.

It is assumed that all smaller size SBus transfers are supported, i.e. if 16 word bursts are supported then 8 and 4 word bursts are also supported. Also, the ATM622-s has the notion of two different devices in the system, each supporting different bus sizes (an example would be host memory and an SBus digitizer card). Every descriptor on the transmit data ring and receive buffer ring has a bit to indicate which bus register to use. The bits of the form "4word\_1" are used when the descriptor bus\_reg bit is set to 1, while the bits of the form "4word\_0" are used when the descriptor bus\_reg bit is 0.

The config register is cleared upon reset.

*par\_en (9):*

Parity Enable bit when set, it activates the parity generation and checking logic for SBus and all subsequent slave and master cycles will use the associated parity function. Parity function operates according to IEEE P1496 SBus specification. Parity error detected during slave write access will cause an SBus error ack to be generated rather than the normal byte or word ack. Parity error detected during Master read cycles will cause the master cycle to be terminated immediately and a master parity error is marked on IO status register.

*skip\_retry (8):*

There are some SBus based systems which have the provision of handling multiple DVMA stream state. If one DVMA cycle is terminated with a retry ack another DVMA stream may request for bus if skip-retry function is enabled. If this provision is not supported by the host, the same DVMA cycle which terminated with retry ack must come back with the same address request before another DVMA stream is allowed on the bus.

*64bit\_1 (7):*

Set if 64 bit (extended transfers) are supported for target 1. If set, it is assumed that 64 bit transfers are supported for all allowed burst sizes.

*16word\_1 (6):*

Set if 16 word (64 byte) transfers are the largest burst size supported for target 1. If this bit is set it is assumed that 8 and 4 word bursts are also supported for target 1. Do not set 4word\_1 and 8word\_1 in addition to this bit, only set this bit.

*8word\_1 (5):*

Set if 8 word (32 byte) transfers are the largest burst size supported for target 1. If this bit is set it is assumed that 4 word bursts are supported, do not set 4word\_1 in addition to this bit.

*4word\_1 (4):*

Set if 4 word (16 byte) transfers are the largest burst size supported for target 1. I. e. only set this bit if the platform does not support 8 or 16 word transfers.

*64bit\_0 (3):*

Set if 64 bit (extended transfers) are supported for target 0. If set, it is assumed that 64 bit transfers are supported for all allowed burst sizes. See 64bit\_1 above for details.

*16word\_0 (2):*

Set if 16 word (64 byte) transfers are the largest burst size supported for target 0. See 16word\_1 above for details.

*8word\_0 (1):*

Set if 8 word (32 byte) transfers are the largest burst size supported for target 0. See 8word\_1 above for details.

*4word\_0 (0):*

Set if 4 word (16 byte) transfers are the largest burst size supported for target 0. See 4word\_1 above for details.



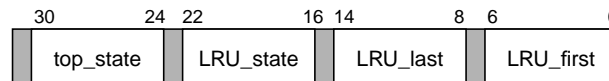
*software\_reset*



If this address is touched (either read or write) the chip will perform an internal reset, and reset all internal fields to poweron state.

### 2.5.1.2 *RX\_LRU and free memory registers*

*LRU\_pointers*



*top\_state (30:24):*

The top-most DMA state actually on the doubly linked list. In most cases this will be 1 (DMA state #0 is the OAM DMA state, initialized off the list). However, if software decides to initialize DMA state #1 off the list this value would be 2. This should be the same as LRU\_first.

*LRU\_state (22:16):*

The current position in the linked list for flushing channels early. Software must initialize this field to 1. If additional DMA states are initialized off of the LRU list, this must point to the first state on the list.

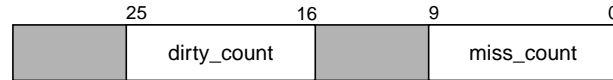
*LRU\_last (14:8)*

Pointer to the last item in the RX DMA state Least Recently Used linked list. This DMA state pointed to by this register is the least recently used state, and is thus the first candidate for reclamation. On poweron this register must be initialized to 0x7F. Once the chip begins operation this field is under the control of the hardware and should not be modified by software.

*LRU\_first (6:0)*

Pointer to the first item in the RX DMA state Least Recently Used linked list. Used as part of the DMA state reclamation policy. On poweron this field must be initialized to 1. Once the chip begins operation this field is under the control of the hardware and should not be modified by software.

*LRU flush counts:*



These registers are present for tuning information, to see how well the RX DMA state cache performs in a real network. The hardware does not use these values for any purpose.

*miss\_count (9:0):*

Counts the number of 'misses' that has occurred as a result of a new arriving cell whose state is not cached in the on-chip RAM. A DMA state read will occur as the result of 'miss'. This only counts misses which do not result in the flush of a dirty DMA state (which are counted below). The total number of DMA state misses is miss\_count + dirty\_count.

*dirty\_count (25:16):*

Counts the number of DMA state misses which had to have a dirty DMA state written out first before the new DMA state was read in.

*first\_free (11:0):*



Pointer to the first bucket in the free list. On power on software must initialize all bucket's next pointers to put them all on the free list, and initialize this register to point to the first bucket on the list. Note that bucket 0 is invalid, the list should begin with bucket 0x002. Once the chip is operating this register is under the control of the hardware and should not be modified by software.

*rf\_rl\_bucket (11:0):*



The first unused bucket, initialized off the list. Software should initialize this value to 0x001, and must set the next pointer for bucket #1 to 0.

*num\_buckets:*



Counts the number of buckets on the free list. Note that under certain conditions this count can be wrong; after a packet crash this count may underestimate the number of buckets actually on the list. The hardware does not use this value for any purpose, it is present only for tuning information so that future chips can be designed with some knowledge of how many buckets are used in real network operation. Software must initialize this value to 2047.

*RX status*



If hardware has been active (in the config register `en_rx == 1`), and software writes to the config register to clear the `en_rx` bit, the hardware will remain active for a short period of time before returning to its IDLE state. After disabling RX software should poll this register until the RX idle bit is 1. This should not take more than a few microseconds.

If hardware is not idle, state gives the DMA state number the unload section is currently working on. Software can look this DMA state up in the LRU list to find out what VCI is being worked on, if desired.

### 2.5.1.3 VCI commands per connection

Software can perform certain functions on a particular connection while the connection operates, by using this command facility. Software can turn a VCI off, switch a VCI to a targeted buffer ring, or switch a VCI to a non-targeted buffer ring.

Turning a VCI off takes effect immediately; the hardware will drop any buffered data for that VCI and return any buffers to the completion ring. Target and non-target commands take effect at the beginning of the next packet, meaning the hardware will finish whatever buffer it was working on and then get the next buffer from the new buffer ring.

To give a command to a particular VCI, software writes a word to an address corresponding to the type of command and the VCI #. The types of commands are:

01 Make VCI targeted.

10 Make VCI non-targeted.

11 Turn VCI off.

The VCI number is the 10 bit channel number, which may be a combination of VC and VP bits.

For the target command, software should write the new targeted buffer descriptor ring to the address for the desired VCI. For the off and nontarget commands software just needs to write a word of garbage to the address for the VCI.

The address is: (ATM\_SYS base address) + (command\_type << 13) + (VCI << 2)

Note that reading back from this address will not necessarily return the same data as was written.

The hardware can handle only one VCI command at a time. Software must make sure the previous command has been completed before issuing another command. To check the status of the previous command software can read from the command register, which is located at address (ATM\_SYS base address) + (command\_type << 13) + (0x4). The format of this register is:

VCI command register:



The hardware stores the address of the most recently issued command in registers while servicing it. If the cmd field is 00, then the hardware is finished servicing the previous command software can issue another. If cmd!= 00, then the hardware is working on a command of type cmd for the given VCI.

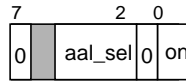
A VCI command is serviced at the highest priority. The hardware will finish whatever cell it was processing and then go service a pending VCI command. If when software reads this register the command is still pending software should spin and poll this address (perhaps in a 256 entry for loop), since the command should be completed shortly.

The status bit indicates if the previous command succeeded or failed. If cmd == 00, then software should look at the status bit. If the status bit is set, the previous command failed. Software should give the command again (software is not re-

quired to try again). The conditions which can cause a command to fail are transient things like not having a free DMA state. It is quite likely that if the command is re-issued a second time it will succeed.

#### 2.5.1.4 Turning on a VCI

To turn on a VCI, software must write the initial DMA state to the `_external_` (noncached) RAM for the VCI number. Then software must do a byte write to the VCI map, and set the entry to noncached, and turned on (0x01). The VCI map contains additional information for non-cached channels, which software must initialize:



#### 2.5.1.5 VCI/VPI bit selection

The ATM622-s can handle up to 1024 connections on receive. It can take all 10 bits of the connection identifier from the VC, or up to 8 bits from the VP (with the remaining bits from VC). The selection of how many bits to take from VP/VC is defined by a global register, VPIbits. The VPIbits setting determines which bits from the ATM header are decoded and used to look up the connection in the VCI map.

**Figure 2-13** VPIbits

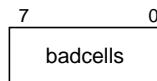
VPIbits	VCI map decoded									
	9								0	
00	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0
80	VP1	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0
C0	VP0	VP1	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0
E0	VP0	VP1	VP2	VC6	VC5	VC4	VC3	VC2	VC1	VC0
F0	VP0	VP1	VP2	VP3	VC5	VC4	VC3	VC2	VC1	VC0
F8	VP0	VP1	VP2	VP3	VP4	VC4	VC3	VC2	VC1	VC0
FC	VP0	VP1	VP2	VP3	VP4	VP5	VC3	VC2	VC1	VC0
FE	VP0	VP1	VP2	VP3	VP4	VP5	VP6	VC2	VC1	VC0
FF	VP0	VP1	VP2	VP3	VP4	VP5	VP6	VP7	VC1	VC0

The important thing to note is that VP bits are “backwards” from their natural positions, as shown in the table.

### 2.5.1.6 Bad cell counter

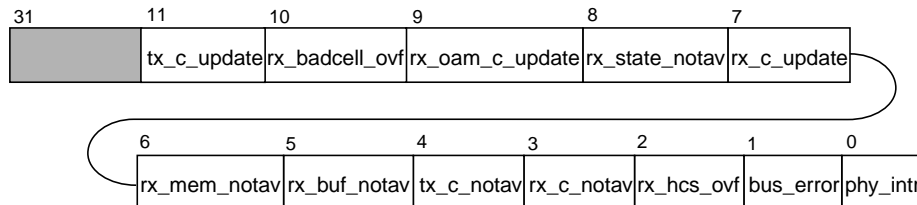
RX discards any cells which arrive for turned off VCIs. However RX counts the number of cells which arrive in an 8 bit counter, and interrupts when that counter overflows. The counter is slave readable, and does not clear on read.

**Figure 2-14** Badcell counter



### 2.5.1.7 Interrupt registers

#### Status



The status register includes an auto clear function upon read for all bits except phy\_intr. The phy\_intr status bit is cleared after the source of interrupt is identified and removed by examining the phy status register. All bits are cleared upon hardware or software reset except phy\_intr. phy\_intr simply reflects the interrupt asserted by phy device.

#### tx\_c\_update (11):

There are two definitions for tx\_c\_update depending on tx\_intr\_mode selected in the configuration register. tx\_intr\_mode =0 indicates that tx interrupt is per tx\_all description and tx\_intr\_mode=1 indicates that tx interrupt is per tx\_individual description.

*tx\_c\_update* for *tx\_all*:

When a tx packet has been transferred to local buffer memory and there is no other packet posted in the corresponding tx channel (sw kick value equal to hw tx descriptor ptr), even if other channels still have packet posted waiting for transmission, *tx\_c\_update* is set in the status register to indicate the last packet transfer done for the corresponding channel.

*tx\_c\_update* for *tx\_individual*:

The *tx\_c\_update* is set on every tx completion ring update.

*rx\_badcell\_ovf* (10):

This bit is set whenever the 8 bit bad-cell counter overflows. Bad cell is defined as cells destined to channels which have not been turned on.

*rx\_oam\_c\_update* (9):

Indicates that an OAM cell has arrived, has been placed on the buffer ring for VCI 0, and has been placed on the completion ring. This interrupt is not delayed as the *rx\_c\_update* is.

*rx\_state\_notav* (8):

Indicates that RX needed to read a new DMA state in to service a cell arrival, but the least recently used DMA state could not be flushed out. This indicates that the ATM622-s is thrashing its cache badly, software should reduce the number of open connections. The chip will continue to operate when this happens, but it will begin to drop incoming cells.

*rx\_c\_update* (7):

Indicates that the ATM622-s has posted to the receive completion ring.

*rx\_mem\_notav* (6):

Indicates that the ATM622-s has run out of local receive memory. This indicates severe latency on the bus, such that currently open connections cannot be sustained.

*rx\_buf\_notav (5):*

Indicates that the ATM622-s tried to post to the non-targeted data buffer ring, and found the OWN bit = 0, meaning the descriptor is still owned by software. This is not a fatal condition, the ATM622-s will still receive packets to targeted buffers (if targeted buffers are allocated); however it is a very serious error and should be corrected (by placing more buffer descriptors in the kernel data ring). The packet which experienced this error is lost.

*tx\_c\_notav (4):*

Indicates that the ATM622-s tried to post to the TX completion ring, and found the OWN bit = 0, meaning the descriptor is still owned by software. Software must correct the condition by putting descriptor entries in the TX completion ring and issuing the tx\_completion\_kick command. The hardware transmit load engine will stop until this error has been corrected.

*rx\_c\_notav (3):*

Indicates that the ATM622-s tried to post to the RX completion ring, and found the OWN bit = 0, meaning the descriptor is still owned by software. Software must correct the condition by putting descriptor entries in the RX completion ring and issuing the rx\_completion\_kick command. The hardware receive unload engine will stop until this error has been corrected.

*rx\_hcs\_ovf (2):*

Indicates that ATM622-s HCS error count register has overflowed. This is generally harmless, however if the register overflows frequently it may indicate a problem with the media.

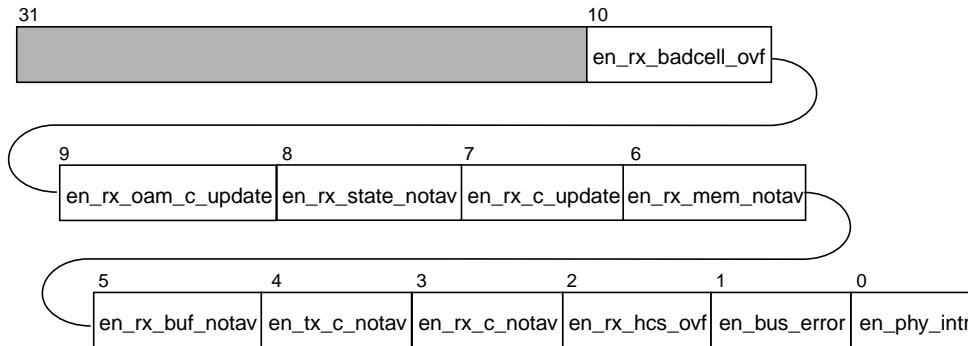
*bus\_error (1):*

Indicates an error was reported by the bus\_interface module. Software must do an additional read from the I/O block interrupt register to determine the exact error. The bus\_error signal is cleared when the I/O interrupt register is read (read status auto clear) or when the source of interrupt is removed.

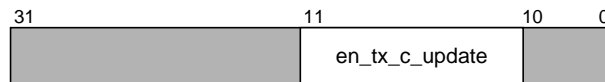
*phy\_intr (0):*

Indicates an interrupt is being generated by the external PHY circuitry. Software must query the PHY for cause. Refer to the manual for the PHY chip for additional information. This bit is cleared when the source of interrupt in the phy module is removed or when the interrupt is masked.



*Interrupt enable register*

There is one bit in this register for each bit in the status register. When the interrupt enable bit is set, it indicates that the corresponding status bit, when set, will cause an interrupt. The interrupt will otherwise be masked for the corresponding status bit. The interrupt enable register is cleared upon reset.

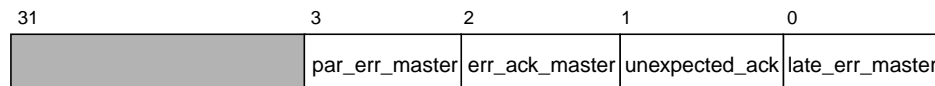
*Tx\_c\_update Interrupt enable register**en\_tx\_c\_update (11):*

If set, tx\_c\_update interrupt is enabled. This enable bit is separated from the rest because software needs to be able to toggle it quickly without acquiring a mutex and doing read-modify-write.

---

**Note:** Reading tx\_c\_update interrupt enable register, or Interrupt enable register returns both registers values. Bits <9:0> will represent Interrupt enable register and bit <10> represent tx\_c\_update interrupt enable register.

---

*I/O status register:*

The ATM622-s contains a separate status register for the IO bus. Any error in the IO bus are propagated to the system side and generate an interrupt as the bus\_error bit in the main status register. When software detects a bus\_error con-

dition it must read the IO specific bus register to determine the exact error. The status register for the SBus IO block is shown here, other buses will be similar but not identical.

*par\_err\_master (3):*

Parity error detected during a master read cycle.

*err\_ack\_master (2):*

Error ack detected during a master cycle.

*unexpected\_ack (1):*

Unexpected SBus Ack received during a master cycle. The master cycle expects to see a word\_ack in 32 bit mode and double-word\_ack in 64 bit mode.

*late\_err\_master (0):*

Late error signal detected during master cycle.

*RX\_intr\_wait:*



*vmin (12:10):*

Specifies the number of packets to be received before signalling another RX\_c\_update interrupt. The hardware interrupts on the first packet to be received, and will then not interrupt again until either the vmin or vtime counters expire. When one of the counters expires, both are reset.

*vtime (9:0):*

Specifies the amount of time to wait from the arrival of a packet until an rx\_c\_update interrupt will be signalled, in units of 12.8 microseconds.

### 2.5.1.8 Tx Status Register

*TX\_status*

31	25	24	19	18	15	13	9	8	5	4	2	0			
Tx_host			Tx_host_data_path			Tx_host_req				Tx_segmentation		Tx_cell_control		Tx_cell_data_path	

This register will be used with the transmit enable signal and as a debug mechanism. The register fields reflect the status of the tx section. The status indications are always changing so if they are to be used functionally, great care should be taken to make sure that the desired status signals will not be missed. The fields will be defined only for known functionality

*Tx\_host (31:25):*

(31:25) = 00 hex idle state

*Tx\_host\_data\_path (24:19):*

(24:19) = 00 hex idle state

*Tx\_host\_req (18:15):*

(18:15) = 0 hex idle state

*Tx\_segmentation (13:9):*

(13:9) = 1d hex or 00 hex idle state

*Tx\_cell\_control (8:5):*

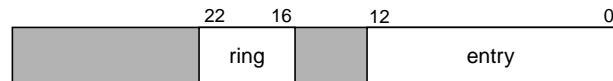
(8:5) = 0 hex idle state

*Tx\_cell\_data\_path (4:2):*

(4:2) = 0 hex idle state

### 2.5.1.9 Tx descriptor kick

#### *TX\_descriptor\_kick*



Software writes all kick commands for all descriptor rings to this register. The hardware can only execute one command at a time, if software issues another while the first is still being dealt with, the SBus ack will be delayed for the second until the first is completed. For best performance, limit the number of kicks to one per packet (i.e. if a packet is queued up as three descriptors, write once to the kick register for the last descriptor, not three times).

#### *ring (22:16):*

This number indicates which of the 127 channels is “kicked”.

#### *entry (12:0):*

The last descriptor entry in the ring currently queued by the software. The entry size is dependent upon the size of the descriptor ring. For ring numbers 8 through 127 the descriptor ring size is 64 and the entry number may range from 0 to 63. For ring numbers 0 through 7 the size of the descriptor ring is programmed in the Tx\_descriptor\_ring\_size register. The Tx\_descriptor\_ring\_size register allows rings 0 through 7 to be from 64 to 8K entries, so the entry number may range from 0 to 8191.

### 2.5.1.10 TX\_descriptor\_ring\_size



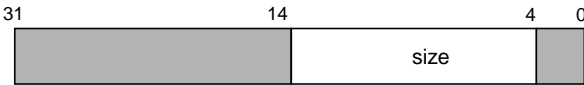
The descriptor ring size register establishes the number of entries on the descriptor ring for rings (or channels) 0 through 7.

**Table 2-3** Tx Description Ring Size Recodings

Size	Number of Entries
0	64
1	128
2	256
3	512
4	1024
5	2048
6	4096
7	8192

**2.5.1.11 Tx Bandwidth Group Table Size, Tx Link Rate and Per-Leaky bucket registers**

*Bandwidth Group Table Size Reg:*



*reserved (31:14):*

Reserved by the hardware for future use.

*size (13:4):*

The number of entries in the Bandwidth Group Table are entered into this field. This number may be from 16 to 16368. This number must be a multiple of 16.

*reserved (3:0):*

Reserved by the hardware for future use.

*Tx Link Rate Reg:*



*reserved (31:9):*

Reserved by the hardware for future use.

*divisor (8:0):*

A cell opportunity occurs every Link Rate. The Link Rate is defined as the 40MHz clock divided by the number in this field. The minimum this number could be is decimal 27 which is used for a link rate of 622Mbits/sec. The maximum this number could be is decimal 512 which corresponds to a Link rate of 33Mbits/sec.

*Leaky bucket reg:*

31	16	15	14	13	9	8	5	4	0
	Leaky bucket On/Off	prescalar	peak	sustain	max_burst				

*Leaky bucket On/Off (16)*

Upon reset (hw or sw), this bit is off (0) and must be set to 1 for the corresponding VBR channel. There are 8 Leaky bucket registers corresponding to 8 VBR channels. Any of the VBR channel can also be used as a CBR channel. If so, this bit must be off.

*prescalar (15:14):*

Selects which clock prescalar to use. The effect of the different encodings is shown in. Valid encodings are:

00: prescalar = 16

01: prescalar = 64

10: prescalar = 256

11: prescalar = 1024

*peak (13:9):*

The peak rate of cell transmission is determined by what is written to the prescalar field and to the peak rate field. It is defined by the following equation where prescalar is equal to 16, 64, 256 or 1024 and the peak number is the decimal value of what is written to the peak field.

The Peak Leaky bucket rate = (40 MHz / prescalar) / peak number

*sustain (8:5):*

The sustain rate of cell transmission is determined by the peak rate and what is written into the sustain field. It is defined by the following equation where the sustain number is the decimal value of what is written to the sustain field.

The Sustain Leaky Bucket rate = peak rate / sustain number

*max\_burst (4:0):*

The maximum burst size (MBS) is the maximum number of cells that may be transmitted in a row at peak rate. The maximum burst size is defined as follows with max\_burst number equal to the decimal value of what is written into the max\_burst field.

Maximum number of cells transmitted in a row at peak rate = 4 \* max\_burst number

**Table 2-4** Setting the Registers

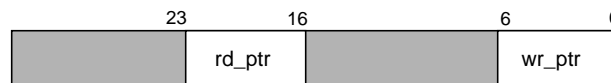
CLK Prescalar	Peak rate	Sustain rate (1/n of peak rate, n = 1-16)	Application
16	Link rate --> 33 Mbps	Down to 2 Mbps+	ATM domain network 622/155 Links
64	265 Mbps --> 8 Mbps (622) Link rate --> 8 Mbps (155)	Down to 500 Kbps	ATM network & FDDI, FastEthernet routers
256	66 Mbps --> 2 Mbps	Down to 125 Kbps	10 Mbps Ethernet routers
1024	17 Mbps --> 518 Kbps	Down to 32 Kbps+	Voice & Video services

---

**Note:** 40 Mhz clk is used to drive the prescalar.

---

### 2.5.1.12 DMA schedule registers

*TX schedule pointers**rd\_ptr (22:16):*

Current location of the load engine in the TX schedule queue. Software must initialize this value to 0 on poweron.

*wr\_ptr (6:0):*

Current location of the TX unload engine in the schedule queue. Software must initialize this value to 0 on poweron.

*RX schedule pointers*



*wr\_ptr (14:8):*

Current location of the RX load engine in the schedule queue. Software must initialize this value to 0 on poweron.

*rd\_ptr (6:0):*

Current location of the unload engine in the RX schedule queue. Software must initialize this value to 0 on poweron.

*Completion kick registers*



*tx\_completion\_kick (9:0):*

The tx completion ring has a fixed size of 1024 descriptors. The software posts empty descriptor to hardware for tx completion updates. The software passes the last empty descriptor posted to the completion ring. The hardware detects the last descriptor available in the completion ring by comparing the bits <15:6> of the tx\_completion\_ring\_ptr (the 10 bit descriptor ptr that last used by the ATM622-s) against the tx\_completion\_kick register. If they are both equal, it would imply that hardware has reached the last descriptor in the completion ring. It otherwise implies that there are one or more descriptors available for hardware to use.





***rx\_completion\_kick (9:0):***

The rx completion ring has a fixed size of 1024 descriptors. The software posts empty descriptor to hardware for rx completion updates. The software passes the last empty descriptor posted to the completion ring. The hardware detects the last descriptor available in the completion ring by comparing the bits <15:6> of the rx\_completion\_ring\_ptr (the 10 bit descriptor ptr that last used by the ATM622-s) against the rx\_completion\_kick register. If they are both equal, it would imply that hardware has reached the last descriptor in the completion ring. It otherwise implies that there are one or more descriptors available for hardware to use.

## 2.5.2 Addressing the RX RAMs

There are two RAMs on RX. The contents of each RAM are as follows:

<b>RAM #1</b>		<b>RAM #2</b>	
DMA state word 1-7	0-1023	VCI map	0 - 255
		LRU list	256-383
next pointers	1024-2047	DMA state word 0	384-511
		Schedule queue	512-543

RAM # 1 contains words 1 through 7 of each of the 128 cached DMA states. Each DMA state is 8 word aligned for ease of addressing. The top word within this 8 is unused, except for address 0 which contains the RX non targeted buffer ring pointer, address 32 (byte addressed) which contains the completion ring pointer, and address 64 which contains the non-IP buffer ring pointer. The 2048 next pointers are also in RAM #1.

RAM # 2 contains the VCI map, LRU list, word 0 from each of the 128 cached DMA states, and the schedule queue.

### 2.5.3 IO Address Map

**Table 2-5** IO Address Map

<b>Device</b>	<b>IO Address</b>	<b>R/W</b>	<b>Width</b>
EPROM	0000000-00FFFFFF	R	8 bits
PHY	0100000-01FFFFFF	R/W	8
TX Bandwidth Group Table	0200000-0200FFF	R/W	32
TX Buffer Mem	0201000-023FFFF	R/W	32
RX Buffer Mem	0300000-0317FFF	R/W	32
RX non-cached states	0318000-033FFFF	R/W	32
TX Control RAM #1	0400000-040181F	R/W	32
RX Control RAM #1	0500000-05FFFFFF	R/W	32
RX Control RAM #2	0600000-06FFFFFF	R/W	32
ATM-SYS	0700000-07FFFFFF	R/W	32
IO	0800000-08FFFFFF	R/W	32
Spare (IDPROM)	0900000-09FFFFFF	R	8

**Table 2-6** ATM-SYS Register address Map

<b><u>Register/IO-command</u></b>	<b><u>IO Address</u></b>	<b><u>R/W</u></b>
<b>General:</b>		
config_reg	0700000	R/W
status_reg (w/ auto clear)	0700004	R
mask_reg (except tx_c_update mask)	0700008	R/W (see Note)
tx_c_update mask register	070000C	R/W (see Note)
rx_hcs_err_reg	0700010	R
rx_intr_wait_reg	0700014	R/W
led_reg	0700018	R/W

**Note:** Reading tx\_c\_update interrupt enable register, or Interrupt enable register returns both registers values. Bits <9:0> will represent Interrupt enable register and bit <10> represent tx\_c\_update interrupt enable register.

<b>Transmit:</b>		
tx_descriptor_kick_reg	0700104	R/W
tx_leaky_bucket_reg0	0700110	R/W
tx_leaky_bucket_reg1	0700114	R/W
tx_leaky_bucket_reg2	0700118	R/W
tx_leaky_bucket_reg3	070011C	R/W
tx_leaky_bucket_reg4	0700120	R/W
tx_leaky_bucket_reg5	0700124	R/W
tx_leaky_bucket_reg6	0700128	R/W
tx_leaky_bucket_reg7	070012C	R/W
tx_completion_kick_reg	0700140	R/W
tx_schedule	0700144	R/W
tx_descriptor_ring_size	0700148	R/W

---

**Transmit:**

---

tx_link_rate	0700180	R/W
tx_bandwidth_group_table_size	0700184	R/W
tx_status	07001C0	R

---



---

**Receive:**

---

RX status register	0700200	R
unload register B (debug only)	0700204	R
unload register (debug only)	0700208	R
unload register D (debug only)	070020C	R
unload register E (debug only)	0700210	R
unload register F (debug only)	0700214	R
unload register G (debug only)	0700218	R
unload register H (debug only)	070021C	R
rx_completion_kick_reg	0700220	RW
schedule queue rd wr ptrs	0700240	RW
schedule queue top/bottom entry	0700244	R
free list first_free	0700280	RW
free list top_bucket	0700284	RW
free list num_free	0700288	RW
load register A (debug only)	07002C0	R
load register B & F (debug only)	07002C4	R
load register C (debug only)	07002C8	R
load register E (debug only)	07002CC	R
load register G (debug only)	07002D0	R
load register H (debug only)	07002D4	R
VPIbits	07002D8	RW
bad cell counter	07002DC	R

---

---

**Receive:**

cache manager top_state, LRU_state LRU_first LRU_last	07002E0	RW
cache manager cm register B (debug only)	07002E4	R
cache manager cm register A (debug only)	07002E8	R
cache manager dirty_count and miss_count	07002EC	R
VCI targeted command base address range 0702000 ->	0706FFC	RW (see Note)

---

**Note:** What is read from the VCI command addresses will not be what was written. Even addresses will return the last word of data written to a command address. Odd addresses will return the VCI number the command applies to and the type of command.

---



---

**IO block:**

software_reset_reg	0800000	W
io_config_reg	0800004	R/W
io_status_reg (w/ auto clear)	0800008	R

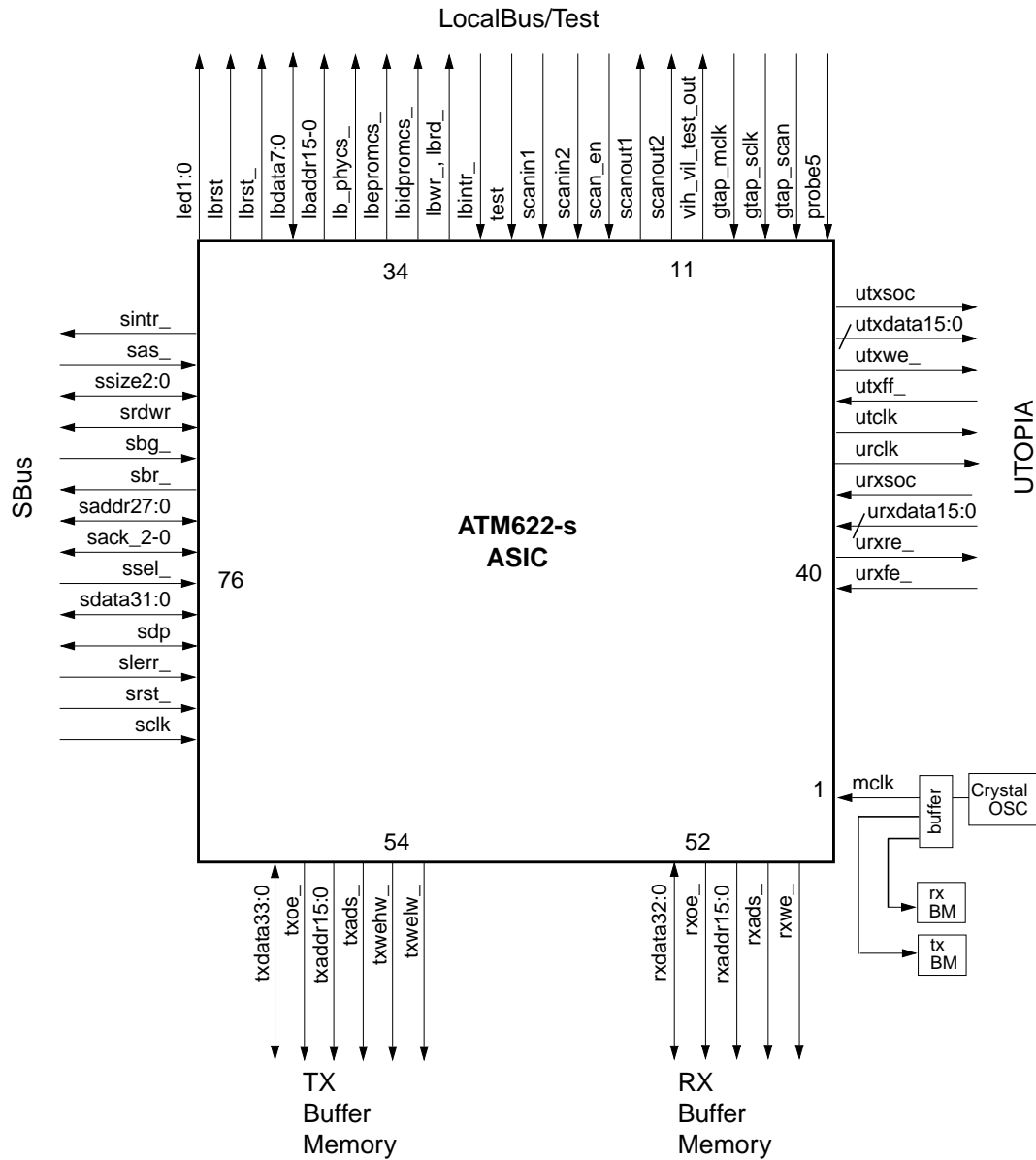
---



### *3.1 Overview*

The ATM622-s is a 0.65 micron CMOS 388 pin BGA with 268 IO signals.

**Figure 3-1** ATM622-s pinouts



Pin Count 76+34+11+40+1+52+54 = 268



### 3. Pin-outs

**Table 3-1** Pin Out Description

Signal Name	Pin #	I/O	Description
TEST	C1	I	1 = test mode 0 = functional mode
LBINTR_	AF23	I	Local Bus Interrupt (active low)
MCLK	N26	I	Master Clock 40 Mhz
SAS_	B25	I	SBUS Address Strobe (active low)
SBG_	D25	I	SBUS Bus Grant (active low)
SCLK	C26	I	SBUS SCLK up to 25 Mhz
SLERR_	C11	I	SBUS Late Error (active low)
SRST_	B3	I	SBUS Reset (active low)
SSEL_	C25	I	SBUS Select (active low)
URXFE_	AF9	I	Utopia Receive FIFO Empty (active low)
URXSOC	AD10	I	Utopia Receive Start of Cell
UTXFF_	AF8	I	Utopia Transmit FIFO Full (active low)
URXDATA15-0	AE10,AF10,AC11, AD11,AE11,AC12, AD12,AE12,AD13, AE13,AF13,AD14, AE14,AF14,AC15, AD15	I	Utopia Receive Data
LED1-0	F24,F23	OD	Signals to drive LED's on card
SPARECS_	F25	O	Spare chip select signal (active low)
RXADS_	H25	O	RX Ext. Buffer Address Strobe (active low)
RXWE_	H26	O	RX Ext. Buffer Write Enable (active low)
RXADDR12	M25	O	RX Ext. Buffer Address bit 12
SBR_	E25	O	SBUS Bus Request (active low)
SINTR_	E24	OD	SBUS Interrupt (active low)
TXADS_	C2	O	Tx Ext. Buffer Address Strobe
TXOE_	P1	O	Tx Ext. Buffer Output Enable
TXWEHW_	D2	O	Tx Ext. Buffer Write Enable bits 33,31-16
TXWELW_	D1	O	Tx Ext. Buffer Write Enable bits 32,15-0

**Table 3-1** Pin Out Description (Continued)

Signal Name	Pin #	I/O	Description
TXADDR15-0	J3,J2,J1,H3 H2,H1,G4,G3 G2,F4,F3,F2, E4,E3,E2,E1	O	Tx Ext. Buffer Address
UTCLK	AD9	O	Utopia Transmit Clock 40 or 20 Mhz
URCLK	AE9	O	Utopia Receive Clock 40 or 20 Mhz
URXRE_	AC10	O	Utopia Receive (active low)
UTXSOC	AE8	O	Utopia Transmit Start of Cell
UTXWE_	AD8	O	Utopia Transmit Write Enable (active low)
UTXDATA15-0	AD2,AE2,AE3,AF3 AD4,AE4,AF4,AC5 AD5,AE5,AF5,AC6 AD6,AE6,AC7,AD7	O	Utopia Transmit Data
LBPHYCS_	AE24	IO	Local Bus Physical Layer Chip Select
LBWR_	AE25	IO	Local Bus Write Enable (active low)
LBRD_	AD25	IO	Local Bus Read Enable (active low)
LBROMCS_	AF24	IO	Local Bus PROM Chip Select (active low)
LBRST	AD23	IO	Local Bus Reset
LBRST_	AE23	IO	Local Bus Reset (active low)
LBDATA7-0	AE15,AC16,AD16, AE16,AC17,AD17, AE17,AF17	IO	Local Bus Data
LBADDR15-0	AD18,AE18,AF18, AD19,AE19,AF19 AC20,AD20,AE20, AC21,AD21,AE21, AC22,AD22,AE22, AF22	IO	Local Bus Address
RXADDR15-13 RXADDR11-0	G25,H24,N24, M24,M23,L25,L24, L23,K26K25,K24, K23,J26,J25,J24	IO	RX Ext. Buffer Address

### 3. Pin-outs

**Table 3-1** Pin Out Description (Continued)

Signal Name	Pin #	I/O	Description
RXDATA32-0	AD26,AC26,AC25, AC24,AB26,AB25, AB24,AB23,AA25, AA24,AA23,Y25, Y24,Y23,W26, W25,W24,V26, V25,V24,U26, U25,U24,U23, T25,T24,T23, R25,R24,R23, P26,P25,P24	IO	RX Ext. Buffer Data
SDP	D3	IO	SBUS Data Parity
SRDWR	C13	IO	SBUS Read Write signal
SACK_2 SACK_1 SACK_0	A5,B7,B9	IO	SBUS Acknowledge Signal
SADDR27-0	A3,A4,B4,C4, B5,C5,D5,B6, C6,D6,C7,D7, A8,B8,C8,A9, C9,A10,B10,C10, D10,B11,D11,B12, C12,D12,A13,B13	IO	SBUS Address

**Table 3-1** Pin Out Description (Continued)

Signal Name	Pin #	I/O	Description
SDATA32-0	B15,C15,D15,B16, C16,D16,A17,B17, C17,D17,A18,B18, C18,A19,B19,C19, B20,C20,D20,B21, C21,D21,A22,B22, C22,D22,A23,B23, C23,A24,B24,D24	IO	SBUS Data
SSIZE2-0	A14,B13,C12	IO	SBUS Size
TXDATA33-0	Y3,Y2,W3,W2, W1,V3,V2,V1, U4,U3,U2,U1, T4,T3,T2,R4, R3,R2,P3,P2, Y4,N3,N2,N1, M4,M3,M2,L4, L3,L2,K4,K3, K2,K1	IO	Tx Ext. Buffer Data

# *Appendix*

---



Errata .....	118
--------------	-----

## Errata

BUG NUMBER: 1

BUG DESCRIPTION: Timing path problem.

COMMENTS: Since the violation is so small it makes sense to wait and see if this problem will show up in a split process lot at the maximums and minimums of voltage and temperature.

BUG NUMBER: 2

BUG DESCRIPTION: RX spent\_partial buckets are not immediately returned to the free list, they remain on the DMA state's linked list until the next packet for that VCI arrives. The bucket is not leaked; the bucket could be returned to the free list for use by someone else much more quickly.

COMMENTS: By itself this is not severe enough to prompt a spin. If we spin for other reasons we should investigate fixing this.

BUG NUMBER: 3

BUG DESCRIPTION: Timing violation of half mclk critical path in IO block.

COMMENT: Fix was performed on board to use healthy clock driver before power on. Duty cycle  $45 \pm 55$ . No fix needed.

BUG NUMBER: 4

BUG DESCRIPTION: (one line description of the bug) perr (parity error) bit of IO status register is inverted in 32 bit mode SBus thus not usable.

COMMENTS: software workaround possibilities: In 32 bit SBus machines, software should either not enable parity check or mask off bus error interrupt or ignore bus error interrupt if parity error is set.

BUG NUMBER: 5

BUG DESCRIPTION: Buffer chaining has race condition in CRC calculation.

COMMENTS: When buffer chaining the CRC might be written back to RAM too early. After getting the next buffer the CRC is always read from RAM. This can result in a CRC error being signalled for a good packet.

BUG NUMBER: 6

BUG DESCRIPTION: Race condition in free list manager involving rf\_rl\_free. When first\_free and rf\_rl\_bucket are both zero, and unload returns a bucket, there are a few clock periods when rl\_rf\_free indicates there are two buckets when there is only one.

COMMENTS: None

BUG NUMBER: 7

BUG DESCRIPTION: Rx VC hangs but no link list corruption. SEEN: Driver (Null AAL case).

COMMENTS: This is an infrequent bug. Every five seconds check DMA state information and verify that if VC is scheduled that the crc and buf\_ptr has changed in the last five seconds. If not, reset.

BUG NUMBER: 8

BUG DESCRIPTION: Link list Corruption due to pkt crash.

COMMENTS: There are many sources of link list corruption, all handled by one workaround. Store a history of num\_bkt, one entry per memnotav. If 8 consecutive num\_bkt entries are identical, reset chip.

BUG NUMBER: 9

BUG DESCRIPTION: Spurious Mem notav.

COMMENTS: No workaround. Simply ignore.

BUG NUMBER: 10

BUG DESCRIPTION: Media\_fifo RX fifo full.

COMMENTS: No workaround needed.

BUG NUMBER: 11

BUG DESCRIPTION: Local Bus Read Data Corruption while TX DMA active.

COMMENTS: To read a local bus device, disable rx and tx in the ATM622-s, wait 300 us, then a read may be performed. Then enable rx and tx in the ATM622-s.

BUG NUMBER: 14

BUG DESCRIPTION: Link down is due to starvation of VC5.

COMMENTS: Rx reassembly (decree) buffer are serviced on a Round Robin priority. Priority changes when buffer is empty. The case of heavy traffic to the highest priority queue, the signal VC (VC5) may not be serviced at an acceptable rate. Change priority scheme to frequently service VC5.







**SUN MICROELECTRONICS**

2550 Garcia Avenue  
Mountain View, CA, USA 94043  
1-800-681-8845  
[www.sun.com/sparc/](http://www.sun.com/sparc/)