VHDL Design of an ATM Switch

by
James Andrew Gilderson
Bachelor of Engineering in Engineering Physics, McMaster University, 1993

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Applied Science in Electrical Engineering

in the Department of
Electrical and Computer Engineering

We accept this thesis as conforming
to the required standard

[redacted]

Dr. Fayez El-Guibaly, Supervisor

[redacted]

Dr. Vijay K. Bhargava, Supervisor

[redacted]

Dr. G. C. Shoja, Outside Member

[redacted]

Dr. Ron Podhorodeski, External Examiner

Supervisors: Dr. Fayez El-Guibaly, Dr. Vijay K. Bhargava

ABSTRACT

The next generation of communication networks will combine all forms of traffic on one universal network infrastructure. The Asynchronous Transfer Mode (ATM) has been proposed for this infrastructure. However, there still exist many unresolved issues in implementing a full ATM network. This thesis examines how to integrate the different cell flows found in ATM networks into one hardware architecture within an ATM switch while providing the necessary functionality to route cells and manage the network resources.

The general requirements of an ATM switch are presented based on ATM standards documents, and these requirements are used to describe an ATM switch as a set of components using a structured VLSI design methodology. This description is used to develop a modular design for an ATM Layer switch that incorporates all three planes of the ATM Protocol Model as well as other necessary functions.

A survey of current ATM switching fabrics is also presented to examine the switching mechanism and how input and output processors can be added to the fabric to provide a full ATM switch implementation.

Based on the modular ATM switch design, the survey of switching fabrics, and other implementation issues, an ATM switch was implemented using high-level VHDL modelling. This implementation provides many of the ATM switch functions while providing for the integration of software-based modules for signalling and management processors. This ATM switch implementation provides a simple architecture for a local area switch using a TDM bus switching fabric in conjunction with combined input-output buffering. The architecture supports eight classes of user cells in addition to signalling, OAM and ILMI cells, each of which are serviced using a priority queuing mechanism at the output ports. Multicasting is supported, as is cell tagging and selective cell discard. The logic of this VHDL

model has been verified through simulation, and the cell flow between the modules within the switch has been examined through a study of bus arbitration mechanisms and contention resolution. This design is highly modular, providing the means for simple design changes and the use of a variety of switching fabrics, and as such is a candidate for implementation as a VLSI chipset.

Examiners

[redacted]

Dr. Fayez El-Guibaly (Electrical and Computer Engineering)

[redacted]

Dr. Vijay K. Bhargava (Electrical and Computer Engineering)

[redacted]

Dr. G. C. Shoja (Computer Science)

[redacted]

Dr. Ron Podhorodeski (Mechanical Engineering)

Table of Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, I would like to thank my supervisors, Dr. Fayez El-Guibaly and Dr. Vijay Bhargava, for providing the necessary guidance for my studies. The atmosphere they furnished was well suited to innovation and the pursuit of excellence. Without their support, both personal and financial, my time at the University of Victoria would not have been such a success.

I would also like to lend my appreciation to my fellow graduate students with whom I interacted during my time at Victoria. I would like to thank them not only for their aid in my own field, but in also providing a forum to discuss and develop skills in other areas of endeavour.

I would also like to thank my family for the times they gave me encouragement, listened to my fears and worries, and gave me the will to continue. Over the years you have provided direction when I was lost. Thank-You!

To
My Family and Friends

List of Acronyms

AAL	ATM Adaptation Layer
ATM	Asynchronous Transfer Mode
B-ISDN	Broadband Integrated Services Digital Network
CA Switch	Campus Area Switch
CAC	Connection Admission Control
CAM	Content Addressable Memory
CBR	Constant Bit Rate
CLP	Cell Loss Priority
CLR	Cell Loss Ratio
CL	Connectionless
CO	Connection Oriented
CO Switch	Central Office Switch
CSF	Cell Switching Fabric
EFCI	Explicit Forward Congestion Indication
FIFO	First In First Out
GCRA	Generic Cell Rate Algorithm
GFC	Generic Flow Control
HEC	Header Error Correction
HOL	Head-of-Line
ILMI	Interim Local Management Interface
IM	Input Module
ISDN	Integrated Services Digital Network

ITU	International Telecommunications Union
LSB	Least Significant Bit
MIB	Management Information Base
MIN	Multistage Interconnection Network
MSB	Most Significant Bit
NNI	Network-Network Interface
NPC	Network Parameter Control
OAM	Operations, Administration and Maintenance
OM	Output Module
PT	Payload Type
QoS	Quality of Service
RTL	Register Transfer Language
SAAL	Signalling ATM Adaptation Layer
SE	Switching Element
SIM	Sub Input Module
SM	System Management
SNMP	Simple Network Management Protocol
SOM	Sub Output Module
STD	Source Traffic Descriptor
TDM	Time Division Multiplexing
UME	UNI Management Entity
UNI	User-Network Interface
UPC	Usage Parameter Control
VBR	Variable Bit Rate
VC	Virtual Channel
VCC	Virtual Channel Connection

VCI	Virtual Channel Identifier
VHDL	VHSIC Hardware Description Language
VLSI	Very Large Scale Integration
VP	Virtual Path
VPI	Virtual Path Identifier

Chapter 1

Introduction

1.1 Motivation

As we move further into the Information Age, better means of communicating become increasingly important. Businesses and individuals are no longer only using the telephone and the fax machine, but are also moving toward video teleconferencing and other computer-based methods of communication. These newer methods, and the ever increasing number of people attempting to use them, have created the requirements for new technologies capable of handling the higher bandwidths that are needed by these applications. By providing one network to carry all forms of traffic, users will only need one interface to support all their communication requirements, and service providers will not need to support several network infrastructures. However, not only must the network be capable of supporting high bandwidths, it must also support several different classes of traffic, each with different performance requirements. The general requirements of these different traffic classes are outlined in Table 1.1, from which it is evident that priority service must be provided both in terms of delay and packet loss. In addition, the network should be capable of supporting both permanent and temporary connections, provide a means to control the rates of the different sources, as well as run under legacy networking equipment[1].

The Asynchronous Transfer Mode (ATM) [1-4] is one technology that has been proposed for the next generation communications network. Though still undergoing the standardization process, ATM has been implemented in both wide and local area networks

1. legacy networking equipment - previously installed networking technology that will not be replaced wholesale with the addition of new equipment

Traffic Type	Bit Rate	Acceptable Bit Error Rate	Acceptable Delay
voice	32 kb/s	10^{-7}	25 ms
video	1.5-15 Mb/s	10^{-7}	1 s
data	1-10 Mb/s	10^{-6}	1 s

Table 1.1. Traffic requirements in a multimedia network [1].

[2]. ATM supports the various traffic classes and bit rate requirements using priority queuing and a variable cell rate, as well as providing a variety of connection types and allowing internetworking with legacy equipment. ATM also supports a wide variety of communication media to adapt to a range of environments including satellite and cellular networks [5-7]. There are still, however, many challenges that exist in implementing a full ATM network, many of which lie in designing a switch that encompasses all the functionality required by the standards [8], as well as leaving the design open so changes to the standards do not require major reworking of the switch architecture.

There exists a large amount of literature that discusses ATM switch architectures [9-19], but the majority only examine the routing of user cells. To implement a full switch, it is also necessary to examine signalling and switch/network management. Current available ATM switches do implement certain aspects of the signalling/management functionality, but there has been little work done on how best to incorporate these entities into the switch hardware to develop a chipset capable of implementing a full ATM switching solution [21].

1.2 Review of Previous Work

The International Telecommunications Union (ITU) introduced the first set of Broadband ISDN(B-ISDN)/ATM standards in 1988 [2]. Further standards were developed through other standard bodies, and the creation of the ATM Forum in 1991 [2] saw the acceleration of the standardization process through the introduction of an industry governed procedure. But even before this, the idea of the small, fast packet switch had been introduced for use in the next generation of high-speed networks [9]. The architecture of these switches has evolved as the requirements of ATM have been further defined through standards bodies like the ITU and the ATM Forum. Both time and space division switches have been proposed [10], each giving different performance in terms of latency and cell loss. Different queuing methods [11, 12] have also been utilized to handle congested states, as well as priority queuing methods meant to handle the various requirements of

the different traffic types [13]. All of these areas deal mainly with the routing/queuing of user cells, and leave open the implementation of signalling and management entities as well as the hardware provision of functions like cell tagging and selective cell discard.

In addition to user cells, both signalling and management cells must be interpreted/ generated/routed. Cell flows must also be monitored, performance data must be gathered and the traffic parameters must be enforced. Implementing these functions has been discussed [20], and networks/chipsets have been designed to incorporate them [21-23], but work still needs to be done on how best to implement them.

1.3 Research Goals

The main research goals of this thesis are summarized as follows:

- develop a functional model for a full implementation of an ATM switch based on standards documents and other reference sources
- develop a flexible hardware framework for this full implementation that provides for interaction with software-based processors as required
- use top-down structured VLSI design methodology to decompose this model into small, easily designed, modular components
- implement the hardware framework as a high-level VHDL model

1.4 Thesis Overview

Following this introduction, Chapter 2 gives some background into ATM, and how ATM switching works. Several aspects of ATM itself are discussed before describing how top-down structured VLSI design methodology can be used to divide the functionality of an ATM switch into small manageable components, each of which performs a small subset of the overall switching requirements.

Chapter 3 discusses many of the proposed ATM switch architectures, and their suitability for both small and large scale switches. The pros and cons of each of the switches are discussed as well as their limitations. In addition, the various forms of queuing that are used in ATM switches are examined, as is the implementation of multicasting in the different types of switches.

Chapter 4 provides an overview of the VHDL implementation detailed in the fol-

lowing chapter. The requirements as outlined in the standards are presented, as are the functions implemented in this switch implementation. This is followed by a description of data flow across the switch.

Chapter 5 presents an implementation of an ATM switch that encompasses user, signalling and management functions. The design presented is a framework for a full switch that would require software implementations of the current signalling and management standards. This design has been implemented as a structural/behavioural model in VHDL, and its logic has been verified through simulation. The design's modular structure makes it suitable for a scalable, VLSI implementation as an ATM chipset.

In Chapter 6 we conclude the thesis by providing a summary of the work, as well as some future directions for study.

Chapter 2

ATM Switch Structure and Functionality

2.1 Introduction

The Asynchronous Transfer Mode (ATM) is a switching/multiplexing technology that was designed to provide networking services over a wide range of applications encompassing all traffic types [2]. Intended as a universal networking technology, ATM is capable of carrying traffic types of several different service classes, each requiring different performance in terms of specified delay and cell loss, in addition to connections of different data rates that are negotiable at connection setup. As a result, an ATM switch must be able to provide several queuing and buffering disciplines to ensure the delay and cell loss characteristics of the different connections.

Data transfer in ATM is performed using short fixed length packets called cells. At an ATM node, a cell is switched according to virtual connection identifiers found in the cell header, that together with the transmission path uniquely identify an ATM connection. These indentifiers have only local significance, and must therefore be updated for use at the next ATM node. A connection-oriented protocol, ATM cells must be transferred in the sequence in which they were transmitted in order for the higher layer message to be recovered at the connection endpoint. The challenge in designing an ATM switch lies in providing sequenced switching of the cells from several ports while maintaining the service quality for each connection. Additionally, the ATM switch must be capable of providing management and signalling functions, including the extraction and handling of non-user cells. By examining these requirements, the functionality of an ATM switch can be divided using an hierarchical VLSI design approach.

2.1.1 The ATM Cell

The basic unit of ATM is the cell, a 53-byte packet consisting of a 5-byte header and 48-byte payload. It is the header that provides the necessary information to intermediate switches for cell routing/handling. The number of fields in the cell header can be 5 or 6 depending on whether the cell is crossing a Network-Network Interface (NNI) or User-Network Interface (UNI) respectively. The format of the ATM cell header at the UNI is shown in Figure 2.1 [8].

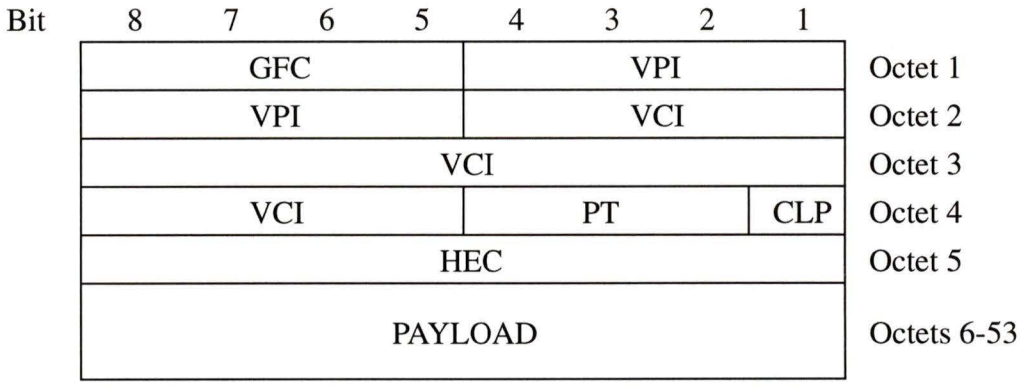


Figure 2.1 ATM cell structure at the UNI [8].

In the header, the use of the 4-bit Generic Flow Control (GFC) at the UNI is currently not defined, and should be set to zero. At the NNI, the GFC becomes the four most significant bits of the VPI field. The Virtual Path and Virtual Channel Identifiers (VPI and VCI) are used to route the cell at a switch, and together must be unique within a transmission path. The 3-bit Payload Type (PT) identifies the type of cell, and can also be used to indicate congestion or used by the higher layer protocols to transmit protocol messages. The Cell Loss Priority (CLP) bit indicates whether the cell is high priority (0) or low priority (1). A low priority cell can be discarded during congestion. The final field is an 8-bit Header Error Correction (HEC) that is used as Cyclic Redundancy Check at each switch to find multi-bit errors or correct single bit errors in the header. The contents of the payload are ignored at intermediate nodes, and depend on higher layers for error detection/correction. The composition of the cell payload is dependent on the service class, as each type of service uses differing amounts of overhead to reconstruct the higher layer message.

2.1.2 ATM Protocol Model

The ATM Protocol Model, depicted in Figure 2.2, shows the bottom three layers of a protocol stack used to transfer information between peer entities[1] across a network [8]. Above these three layers lies higher level protocols like TCP/IP, or DS1/DS3 Constant Bit Rate (CBR) services. The ATM stack segments the higher layer messages into cells, adding additional overhead depending on the service class, and routes the cells across the network using a physical medium.

From Figure 2.2, the ATM Protocol Model is divided into 3 Layers and 3 Planes. The Layers are the Physical Layer, the ATM Layer and the ATM Adaptation Layer (AAL). The planes are the User, Control and Management Planes.

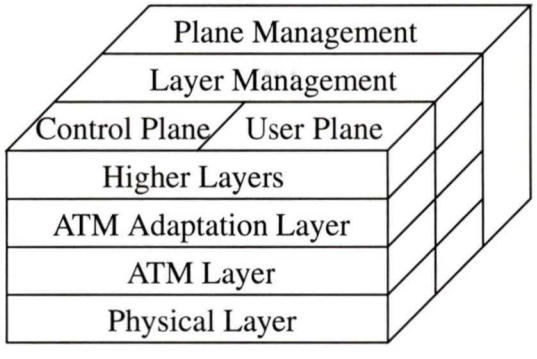


Figure 2.2 ATM protocol model [8].

Plane-wise, the User Plane is responsible for supporting the transfer of user information across the network. The Control Plane handles the negotiation of new connections, the release of established connections, and any other functions relating to connection control. The Management Plane is responsible for management functions including operations like failure notification and connectivity verification. In addition, the management plane is further subdivided into Layer and Plane Management, with Layer Management providing layer-specific management functions, and Plane Management providing functions that span all three layers of the stack.

---

1. peer entities - networking equipment that lies on the same layer of the protocol model

Layer-wise, the Physical Layer handles functions like Header Error Correction (HEC) for ATM cells, cell delineation and cell scrambling/descrambling in addition to providing access to the physical medium. The ATM Layer is responsible for cell multiplexing/switching, header generation, and VPI/VCI translation. The AAL is used to pass protocol data units (PDU's) between the ATM and higher layers in order to make the ATM Layer invisible to the higher layers. The different service classes, connection-oriented (CO) or connectionless (CL) and CBR or Variable Bit Rate (VBR), are also defined at this layer.

In an ATM switch, all three planes must be implemented, though the number of layers implemented depends on the particular plane. In the User Plane, higher layer messages are only recovered at the endpoints, therefore, only the Physical and ATM Layers are required at the switch. For the Control Plane, signalling messages must be recovered using the Signalling AAL (SAAL). As decisions about connection acceptance/rejection may be made at each switch, all three layers must be provided in an ATM entity. If the SAAL is not implemented, then signalling decisions must be made on a network-wide basis. There are currently two standardized Layer Management messages that must be handled by an ATM interface at the UNI [8]. These are Operation, Administration and Maintenance (OAM) cells and Interim Layer Management Interface (ILMI) messages. OAM cells are currently defined for Fault Management, Performance Management, and Activation/Deactivation. ILMI messages are used for the exchange of management information between two UNI Management Entities (UMEs) across a UNI. As ILMI messages can span several cells, they are usually transferred using the AAL5 [8] service class, and must be recovered at the switch, which therefore requires an AAL in the Management Plane.

2.1.3 The Virtual Path and Virtual Channel

The connection identifier of an ATM cell is divided into the VPI and VCI to provide for a two layer switching capability. An ATM switch can be either a Virtual Path (VP) or Virtual Channel (VC) switch for a connection. A VP switch or VP handler routes the cell to the correct output port based solely on the VPI of the cell, translating/rewriting only the VPI. A VC handler switches the cell based upon both the VPI and VCI, translating/rewriting both the VPI and VCI in the process [3].

The Virtual Channel (VC) is the lower level of the switching hierarchy. A Virtual Channel Link or Segment exists between two ATM devices[2] where the VCI is translated or terminated (passed to/from AAL). The end-to-end VC Connection (VCC) exists between the two endpoints where the ATM Layer interacts with the higher layers.

The higher level of the switching hierarchy is the Virtual Path (VP), which is used to bundle several VCs. A Virtual Path Link or Segment exists between two ATM devices where the VPI is translated, which is between any two ATM devices as the VPI is translated at both types of switches, and at the origin/endpoint of the user connection. The end-to-end VP Connection (VPC) exists between two ATM devices where the VCI is originated/translated. Therefore a VPC will only span several nodes when the intermediate nodes are VP handlers. Those VCs that lie within a specific VP by default all possess the same service quality, which is the service quality of the most stringent connection in the path.

2.1.4 ATM Switching

ATM switching is based upon the VPI/VCI and PT. To make cell routing easier, certain predefined headers exist that indicate the type of cell, a subset of which are shown in Table 2.1 and Table 2.2 [8]. The signalling header included is the default one intended for user signalling with the local exchange. The OAM cells defined in Table 2.1 are those for the VP, and have the same VP as the user connection, but a special VCI. The ILMI header is the default one, but in general this header must be configurable. For PT, the user cells can indicate whether congestion was experienced across the network, called Explicit Forward Congestion Indication (EFCI). It can also be used by the AAL, and is currently used by the AAL5 service class [8] to indicate the last cell in a message. The VC-level OAM cells are also identified with the PT, as they have both the same VPI and VCI as the user connection. The ATM Forum has also defined headers for Broadcast signalling and Meta-signalling, though the implementation described in Chapter 4 only implements the required default point-to-point signalling.

From the pre-defined ATM headers listed in Table 2.1 [8], a decision can be made as to whether the cell is user, signalling, management or unassigned, where an unassigned cell is one that has been inserted to maintain the bit rate of a synchronous physical

2. ATM device - a switch, cross-connect or host that supports the ATM protocol stack

medium. Each physical transmission path interfacing with the switch can carry multiple Virtual Paths, each carrying many Virtual Channels. The responsibility of the ATM switch is to route the cells to the correct port, and multiplex the cells from the different connections onto the medium in the correct sequence with the correct header while maintaining the negotiated Quality of Service (QoS) of the connections.

Type	Octet 1	Octet 2	Octet 3	Octet 4
Unassigned	00000000	00000000	00000000	0000xxx0
Signalling	00000000	00000000	00000000	01010aac
Segment VP OAM	0000aaaa	aaaa0000	00000000	00110a0a
End-to-End VP OAM	0000aaaa	aaaa0000	00000000	01000a0a
ILMI	00000000	00000000	00000001	0000aaa0

Table 2.1. ATM pre-defined headers [8].

"a" indicates the bit is available for ATM functions
"x" is a don't care bit
"c" means sender sets CLP = 0, but may be changed in network

PT	Meaning
000	User data cell, congestion not experienced, SDU-type = 0
001	User data cell, congestion not experienced, SDU-type = 1
010	User data cell, congestion experienced, SDU-type = 0
011	User data cell, congestion experienced, SDU-type = 1
100	Segment VC OAM
101	End-to-End VC OAM

Table 2.2. Defined payload types [8].

It is this QoS that dictates the maximum end-to-end delay of the connection, as well as the acceptable Cell Loss Ratio (CLR). The delay of a connection is a function of the queuing delay of the cells across the network. This delay can be reduced for time sensitive traffic, like real-time voice or video, using priority queuing where the delay sensitive traffic is given priority over less sensitive traffic like that of file transfers. Because of the low error rate of modern physical media, the majority of cell loss in ATM networks will be due to buffer overflow in congested switches. A switch under a heavy load will quickly run out of buffer space for an output port. To service connections with low CLR, it is therefore necessary to provide large buffers, and make careful connection admission decisions to limit the probability of a heavily loaded port. Additionally, the traffic contract established at connection setup must be enforced.

2.2 Structured VLSI Design

ATM at the lower two layers of the protocol model is generally a hardware technology [2]. As such it can be implemented using current VLSI methods, and a good design will use a structured approach that uses systematic methods to simplify the design process. The advantages of structured design include allowing for easier management of the design complexity, making design changes easy to implement, and simplifying the verification/debugging process. For these reasons, a structured VLSI design strategy [24, 25] was used in designing the switch proposed here. This strategy consists of four parts: hierarchy, modularity, regularity and information hiding.

2.2.1 Hierarchy

Hierarchy uses the idea of submodules to divide the functionality of the design into simple manageable steps. The top level consists of the main module with its input and output ports. Below this, the next layer maps the port to submodules that are interconnected to form the higher layer component. This continues until the submodules consist of simple functional units. For this approach to work, the interfaces to each module must be well defined and fully specified so the submodules can be easily integrated together.

Languages like VHDL [26] do this through constructs that use lower level components. Each of these components perform a smaller set of the logic required by the higher layer entity. Signals interfacing with the components may correspond to ports of the entity, or be internal signals between the components where port mapping is used to route the signals among the components.

2.2.2 Modularity

Modularity is the use of a certain module several times within the same design, or the same module in different designs. The advantage of this strategy lies in the reduced design time, the need to verify the module only once, and allowing for easier design changes.

For hardware languages, this can be accomplished through the use of design libraries containing often-used components. These components can include basic building blocks like registers and counters, or more complex ones like those that would be found in the interface ports of a switch.

2.2.3 Regularity

Methods to achieve regularity in a VLSI design include data communications, module shape and layout, and routing. Regular data communications requires a consistent pattern. This pattern could be output data available on one edge of the clock, and input data stable on the other edge, or a regular set of flag lines for asynchronous communication. Shape and layout regularity refer to the size of the modules and where their physical interconnections for the different signals are made. I/O ports could be placed on the left and right of the module with control signals applied to the top for example. This sort of regularity simplifies the placement and wire routing problems of VLSI fabrication. Routing regularity is the regular assignment of layers for wiring.

In a software design, regularity can be used in the data communications and the layout. A clocked system can activate subroutines, set flags, and latch data on clock edges. For layout, a structural design uses mapping of signals for interconnects. By defining component interfaces regularly, with clocks followed by control signals, followed by I/O ports for instance, port mapping can be easily accomplished.

2.2.4 Information Hiding

Information hiding involves masking implementation details outside of a module. The internal construction of the module is hidden from the rest of the circuit. In some cases information hiding helps higher level modelling being used to determine the performance of the system.

In terms of hardware description languages, this can mean the use of behavioural constructs that may not be synthesizable, but allow the high-level logic of the system to be verified. This approach speeds the simulation time, and allows for logical errors to be caught at a higher level.

2.3 Description of Switch Design

The following sections outline a generic description of a layered design of an ATM switch. This description does not apply to a particular implementation, but instead deals with the layers of the switch that are used in the VHDL implementation of Chapter 5, data flow across the switch, and how structured VLSI design can be used in the design process.

2.3.1 Layer 1: ATM Layer Switch

Using a hierarchial VLSI design, the top layer of the switch, shown in Figure 2.3, is the black box **`ATM Layer Switch`** that has N input and N output serial data lines, where N is the number of ports in the switch. As this switch lies above the Physical Layer, the serial data lines feed a bit-stream corresponding to a cell, where the first bit of the stream is the MSB of the GFC/VPI. Each of the data lines has an accompanying control line that corresponds to primitive signals from the Physical Layer. From ATM Forum standards, the ATM Layer must receive a cell for every PHY-UNITDATA.indicate signal and pass a cell to the Physical Layer for every PHY-UNITDATA.request signal. As a synchronous physical medium is assumed, an unassigned cell may be passed if no valid cell is queued at an output port. Additional ports may be required for on site access, and direct system management functions, though these are not examined in this project.

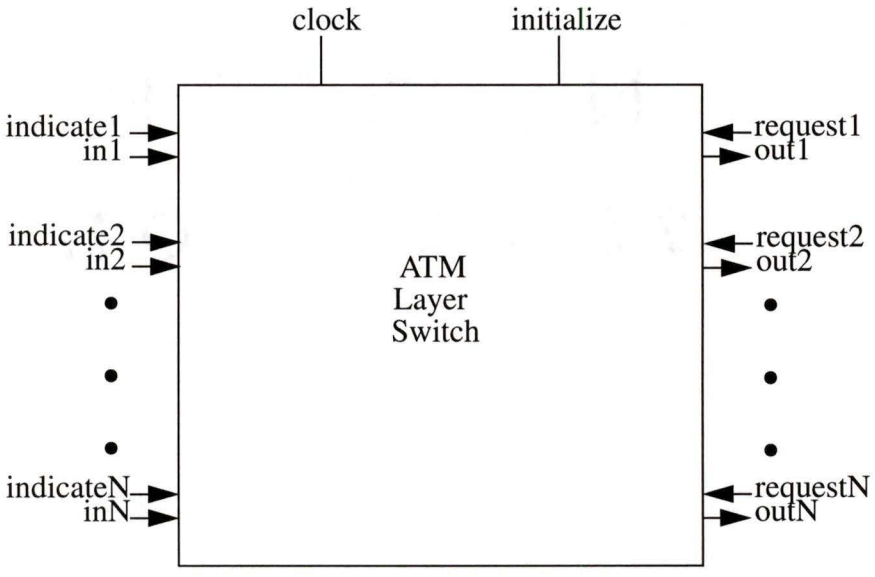


Figure 2.3 ATM layer switch.

2.3.2 Layer 2: Structural Architecture

Figure 2.4 shows the structural architecture found below the **`ATM Layer Switch`**. This layer is composed of five components, the **`Input Module`** (**`IM`**), **`Cell Switching Fabric`** (**`CSF`**), **`Connection Admission Control`** (**`CAC`**), **`System Management`** (**`SM`**), and **`Output Module`** (**`OM`**). The **`IM`** is used for initial processing and rout-

ing decisions based on header discrimination. From the **IM**, user cells along with some OAM cells are fed to the **CSF**, signalling cells go to the **CAC**, and management cell to the **SM** module. Each port has its own bus to the **CSF**, but a common **SM** bus and **CAC** bus are used for the other cell types. The **OM** module sends cells to the Physical Layer and provides such functions as priority queuing, and buffering in case of congestion. The **OM** is also responsible for generating unassigned cells for the synchronous physical media. Both the **SM** and **CAC** can write to the **OM** through their respective buses.

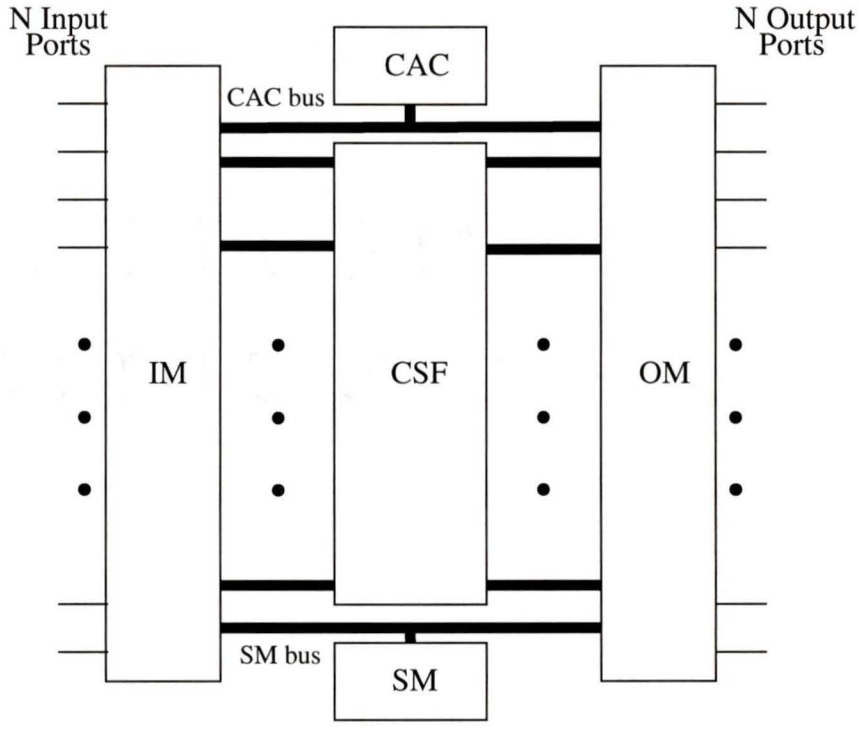


Figure 2.4 ATM Switch Structural Architecture

2.3.3 Layer 3: Modular Units

At this layer lies the repetitive functions that are provided to each physical port. The **IM** and **OM** are further subdivided into identical components, the **Sub Input Module** (**SIM**) and **Sub Output Module** (**SOM**), that provide common cell processing as shown in Figure 2.5. Each of these modules is provided with a unique address to identify an individual port to the **CAC**, **CSF**, and **SM**. There are no VHDL "internal signals" within the **IM** or **OM** as all signals from the **SIM** and **SOM** map to input and output ports of the higher layer module. In a switch implementation, both the **SIM** and **SOM** would lie on the same card that interfaces with the physical medium.

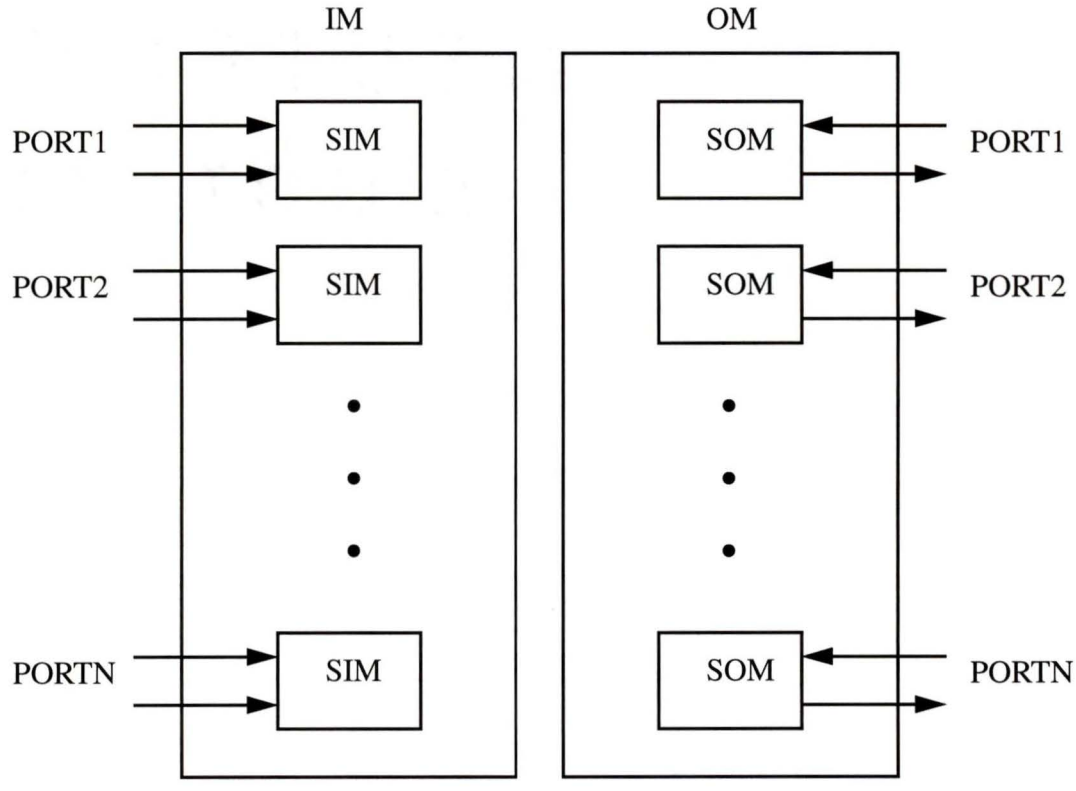


Figure 2.5 Modular units composing the input and output modules.

2.3.4 Layer 4: Functional Units

At this layer, the components are divided into small, functional units that perform one operation, or a small set of operations. The operations are performed on an algorithmic basis, which when verified may be synthesized into a VLSI circuit. One difficulty lies in the fact that much of the functionality has yet to be standardized or is subject to change. So despite the desire for a hardware solution, much of the operations, especially in the management and control plane must be implemented in software, as has been recommended in current standards [8].

2.3.5 Sub Input Module

The **SIM** performs initial processing on the ATM cell, and can be divided into four functional units, as shown in Figure 2.6. This description deals more with the cell flow rather than an actual implementation, which is described in Chapters 4 and 5.

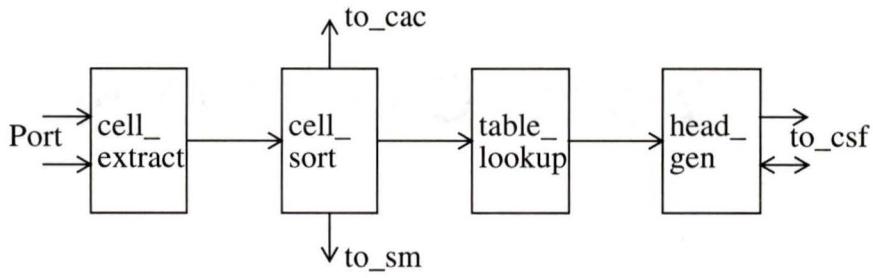


Figure 2.6 Block diagram of Sub Input Module.

The **cell_extract** module removes unassigned cells from the cell stream to prevent unnecessary processing, and this is done through predefined header discrimination. This function is known as cell rate decoupling.

Once cell rate decoupling is performed, valid ATM cells are separated into their different types for processing in the **cell_sort** module. User cells are forwarded to the **CSF**, signalling cells to the **CAC**, and management cells to the **SM**. It has been proposed that some of the **CAC** and **SM** functionality be provided on a port basis [20], and these functions would also be implemented in the **SIM**.

The next step of the **SIM** is to process user cells. These cells must be provided with new headers for the next switch, and this is done through the **table_lookup** module that returns the new header corresponding to the incoming cell's header and port. A routing tag may also be appended to aid in routing the cell across the **CSF**, and to indicate the priority class for the queuing discipline. The tag will be stripped at the **SOM**, so it may be of any form desired by the designer. The table itself may be global or local, each option having its own advantages/disadvantages. A local table reduces the load on the table which lowers the required memory speed, but it also requires a more complex switch with the ability to control entries to many different tables as opposed to the global table with simpler control mechanisms but higher required bandwidth.

Once the new header is obtained, the cell header is rewritten in the **head_gen** module. If the switch is a VP handler, only the VPI is rewritten, while a VC handler

rewrites both the VPI and VCI. The PT, CLP and HEC are left unchanged as the new HEC will be generated at the Physical Layer. At this point the cell is forwarded to the CSF.

2.3.6 Cell Switching Fabric

The switching fabric itself is an area where a large amount of work has been done. A great number of switching fabrics have been proposed that offer different performance in terms of blocking, delay and complexity, and several papers have been written discussing the pros and cons of the various approaches and their suitability for ATM. This area will be further discussed in Chapter 3, but a very short introduction is given here.

The switch fabric routes user cells from one input port to the correct output port. This can be done by either using a switching fabric with a speed equal to the sum of the input rates and serving them sequentially (time-division), or by using some sort of interconnection scheme that provides many routes across the fabric, allowing several cells to be written to distinct output ports concurrently (space-division). The differences between these two approaches include their suitability for large switches, the probability of blocking, and their relative complexity, among other issues.

2.3.7 Connection Admission Control

In general, the steps of the **CAC** can be divided into recovery, interpretation, processing, and generation. ATM signalling has been standardized in the form of ITU standard Q.2931, which outlines signalling messages used to establish and tear-down ATM connections. These messages use the Signalling AAL (SAAL) as the bridge between the ATM Layer and Q.2931. The messages themselves are encapsulated using the AAL 5 service class.

When a signalling cell is received at the **SIM**, it is forwarded to the **CAC**, and stored until the entire message is received, the last cell of which is indicated by the PT. The signalling message is then recovered, and passed to Q.2931 for processing. The messages themselves can be a request to establish a connection, a request to release a connection, or a request to add a party to a multipoint call, once a point-to-point connection has been established, among other messages. The response to these messages can be acceptance or rejection. An acceptance of a connection requires a signalling message to be forwarded to the next node, and a rejection requires the return of a clearing message. The decision to accept or reject is based on the load of the switch, and whether the switch can

accept the call and still provide the QoS for all established connections. The traffic parameters found in the Source Traffic Descriptor (STD) are used to make this decision along with a currently non-standardized algorithm.

Though ATM is intended as a fast, hardware based technology, it is believed that the AAL Layer and above will be implemented in software. Many of the algorithms used for **CAC** decisions require calculations of equivalent bandwidths and similar measures for VBR connections, a task more easily implemented in software. As a result, one of the few hardware processes required at the **CAC** will be bus arbitration to control the write and read operations of the **SIM** and **SOM**.

2.3.8 System Management

As previously discussed, an ATM switch must provide both ATM Layer OAM management, as well as ILMI, provided through the Simple Network Management Protocol (SNMP).

The ITU standard I.610 discusses several requirements for a full OAM implementation, The ATM Forum has chosen to standardize a subset of these in the UNI 3.1 [8]. These are Alarm Surveillance, Connectivity Verification and Invalid VPI/VCI detection. Of the three, the first two use OAM cells, while the third is a message from the SIM to the Layer Management. OAM cells can be either at the VP or VC level and have end-to-end or segment significance. VP OAM cells have the same VPI as the user cells, but a pre-defined VCI. VC OAM cells are identified by the PT, with the same VPI/VCI as the user cells. OAM cells must not be passed beyond their area of relevance, and must be removed or returned at the endpoints. At intermediate nodes the OAM cells may be monitored, but must be passed unmodified. ATM OAM management is another area where the ATM Forum currently recommends software implementation.

The ILMI must also be implemented in the SM. The ILMI maintains a table of relevant ATM and Physical Layer parameters, called the Management Information Base (MIB). Currently ILMI must be supported on ports supporting a UNI. The requirements of the ILMI are the recovery of SNMP messages through the use of AAL 5, and the generation and transmission of return SNMP messages. The MIB may be global to the switch, or provided on a port basis.

As a result of these two management functions, there are two separate streams for management cells, **man_ilmi** and **man_oam**. Again, one of the main hardware require-

ments of this area is the arbitration of the SM bus.

2.3.9 Sub Output Module

The **SOM** receives cells from the **CAC**, **CSF** and **SM**, and they are processed as shown in Figure 2.7. Cells entering **queue_feed** will have any routing tag removed, and are written to the correct queue in **cell_buffer**. If priority queuing is implemented there will be several queues within **cell_buffer**, and each cell will have an associated priority class. The buffers implement a FIFO queue for each of these classes that maintains the cell sequence, but may also use the CLP to remove cells if congestion occurs. At this point Explicit Forward Congestion Indication (EFCI) can be used, if the buffer occupancy passes a threshold, through rewriting the PT. The **queue_server** services the queue according to the requirements of their QoS, and generates an unassigned cell if all the buffers are empty when a request signal is issued.

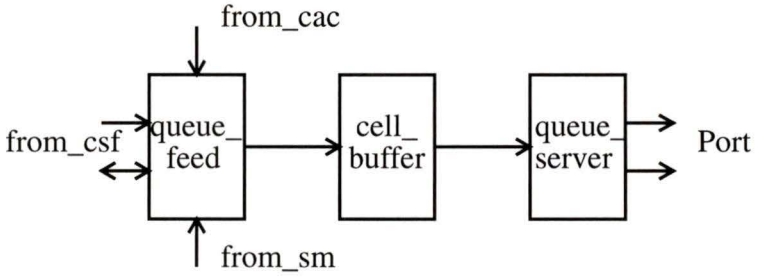


Figure 2.7 Block diagram of Sub Output Module.

2.4 Implementation Issues

For the implementation of a full ATM switch, there are certain other areas that must be designed, and some issues that must be explored. These include traffic control, multicasting, and global vs. local implementation.

2.4.1 Traffic Control

User connections are expected to maintain the traffic parameters as negotiated with the STD. The STD includes such parameters as the peak and sustainable (average) cell rate for both the CLP=0 and CLP=1 cell streams. The network will therefore monitor the connection at the UNI or NNI using the User/Network Parameter Control (UPC/NPC). These modules monitor the cell stream and decide if the connection is compliant to the

negotiated STD. A compliant connection is a connection that conforms to the STD within certain limits[8]. A non-compliant connection risks having its cells either tagged low priority with the CLP bit, or discarded. The traffic control algorithm itself must conform to the results of the Generic Cell Rate Algorithm (GCRA) as defined by the ATM Forum.

2.4.2 Multicasting and Broadcasting

In the ATM Forum standards, a procedure exists for establishing a point-to-multipoint call through UNI signalling. Even multipoint-to-multipoint connections can be established using multiple point-to-multipoint connections. This creates a requirement to replicate user cells at ATM nodes. Methods to accomplish this include Multicast Servers that operate outside the ATM switch, and to which all such cells are forwarded for replication, and the use of replicators within the switch itself in the form of broadcast buses, and generators within an interconnection switching network.

2.4.3 Global vs. Local Implementation

Within an ATM switch, there are certain functions that can be provided on a port or switch basis [20]. As was previously discussed, these include the routing table and certain functions of the **CAC** and **SM**. A switch-wide implementation may introduce a bottleneck when several of the ports are competing for the same resource, but a port implementation replicates functions that may need knowledge of the conditions of the switch as a whole. The best result may lie in some combination of the two. **CAC** decisions require knowledge of the overall load of the switch and possibly knowledge of certain nodes within the fabric, and as a result may best be left to a switch-wide implementation. Much of the management requirements however can be dealt with on a port basis, as they deal with messages on a connection basis, not on a switch-wide basis.

2.5 Summary

Using a structured VLSI design approach, the functionality required by an ATM switch has been broken into small manageable modules that perform a small set of operations. The operations have been divided into five modules that have a well-defined set of responsibilities based on the type of cell. From this, an ATM switch implementation can be created, beginning with a behavioural model of the switch using VHDL.

Chapter 3

Switching Fabric Design Options

3.1 Introduction

The cell switching fabric is the most important element of any ATM switching system. The main goal of a network is to route user information between two endpoints. All other elements of the switch including management and signalling are there to support this primary function. As a result, the choice for the switching fabric becomes very important. An ideal switching fabric will be capable of the loss-free transfer of a cell between an input port and an output port with minimal delay. In addition, it should be expandable over a wide number of ports, provide cell buffering when congested and also provide the means for multicasting. The fabric should be capable of providing this over a wide range of loads and traffic distributions.

This chapter outlines some general switching theory including different switching techniques and buffering methods. This is followed by an overview of ATM switching fabrics that have been implemented or proposed in the literature, and a comparison of the different switching approaches.

3.2 General Switch Design Options

A packet switch is responsible for routing a packet from an input port to a corresponding output port. As a switch will have more than one input port, it must also be capable of resolving contention that arises when two or more input ports are competing for the same resources. These resources may be a particular output port, or a link within the switching fabric itself. The methods that a switch uses to handle routing and contention functions differentiate the different approaches of packet switching [1,2].

3.2.1 Switching Techniques

Most of the architectures for ATM switch fabrics that have been proposed or implemented can generally be classified into two switching techniques: time-division or space-division, generic examples of which are depicted in Figure 3.1.

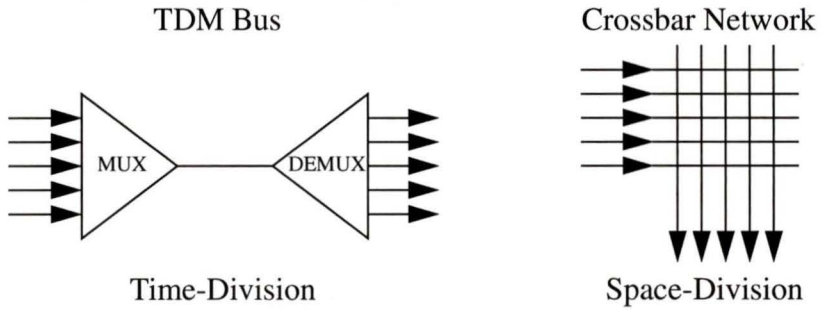


Figure 3.1 Two extreme switching techniques.

The basis of a time-division switch is the sharing of a communication medium among the input ports based on some arbitration/multiplexing scheme. This scheme can be round-robin in nature or prioritized if ports are carrying cells of different priority. To avoid congestion at the input ports, the medium must be capable of servicing all of the input ports during one switching cycle, where a cycle is the time to transfer a cell at the cell transfer capacity of the physical transmission medium. These requirements therefore limit both the capacity of the switch and the maximum number of ports that can be supported as the shared resource's rate must equal the sum of the input port rates. Examples of this switching technique include shared media switches like the time division multiplexed (TDM) bus or ring, and the shared memory switch.

A space-division switch on the other hand provides one or more paths between the input and output ports. These paths run concurrently, as opposed to the sequential operation of the time-division switch. As a result, the switching fabric does not need to operate at speeds higher than those of the input ports. The capacity of this sort of switch is therefore limited by physical implementation restrictions which include device pinout, connector restrictions, and synchronization [10]. Many space-division switches are interconnection networks based on small switching elements (SE). An SE switches a subset of the input port cells, and can be interconnected in stages to connect an input port with any output port. Examples of space-division switches include the crossbar and the

banyan switch [17].

3.2.2 Blocking vs. Non-blocking Switches

A switch is said to be blocking when two or more input ports cannot forward cells to distinct output ports through the switching fabric [1]. Blocking is caused by internal CSF contention for a link. Methods to resolve blocking can include buffering at the contention points within the switching fabric, multiple paths between SE, sorting the packets before switching to avoid link contention, and increasing the bandwidth of links between SE [10].

3.2.3 Cell Buffering

When multiple input ports are competing for the same output port within one switching cycle, cell buffering is needed to prevent cell loss. Cell buffering within a switch can take on several forms, two of which are shown in Figure 3.2.

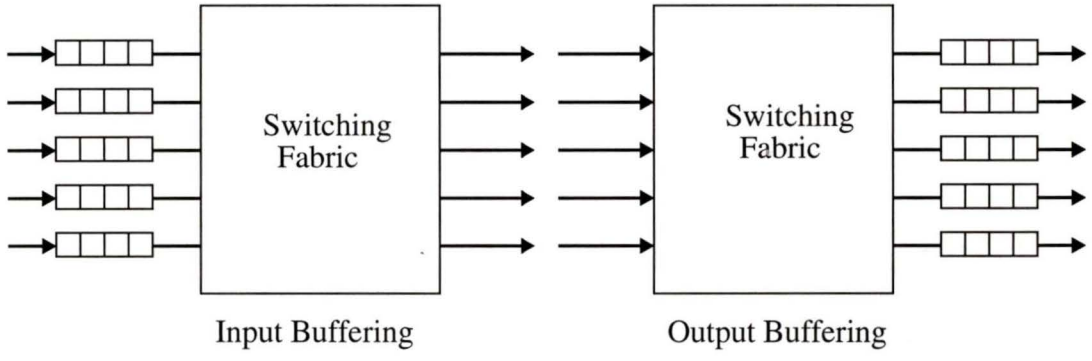


Figure 3.2 Buffering methods.

In an input buffered switch, cells are queued between the input port and the switching fabric. The main difficulty in using input queuing to solve output port contention lies in Head-of-Line (HOL) blocking. HOL blocking occurs when the first cell in the queue cannot be switched but a cell further back in the queue can. There are methods to solve HOL blocking including using a queue window that examines cells to a certain depth in the queue to find a contention-free cell, and speeding up the switching fabric so several

cells from a queue can be switched in one cycle, thus increasing the probability that the head cell can be switched. The difficulty with these methods is that they generally introduce complex arbitration protocols, but without these protocols the maximum supportable offered load of an input buffer fixed-length packet switch is about 58% [11].

Output buffering, where cells are queued between the switching fabric and the output port, eliminates HOL blocking but at the cost of requiring an output port to accept a cell from every input port within a switching cycle. As this may be impossible for large switches, output buffering can be used with a certain cell loss probability. In [9], the cell loss probability for an output buffered switch capable of accepting 8 cells per switching cycle was given as $<10^{-6}$ for a load of 90% under uniform random traffic.

Shared buffering provides a means to decrease the number of buffers required at an input/output port and maintain a good QoS[12]. If fixed length queues are provided for each port or for each service class within a port, many of the queues may be virtually empty while one overflows. Dynamically allocating the available buffers among the queues reduces the probability of cell loss for a fixed number of buffers. The number of required buffers is also reduced, especially for a large number of ports [12].

There may also be internal buffering used in a switch, where cells are stored within the switching fabric itself. This buffering method allows for large-scale switches, but makes it difficult to maintain priority queuing and multicasting [2]. As a result it is not commonly used within a switch.

In general, a combination of the different buffering types can be used within the same ATM switch [2]. While output queuing provides the best performance in terms of throughput/delay as compared to internal and input buffering, internal buffering can be used to reduce the probability of blocking within the switching fabric, and input queuing can be used if the number of cells accepted by an output port is limited, or control/management cells are generated at the input port. Shared buffering provides the best performance in terms of the number of required buffers, but requires more complex control logic.

3.2.4 Expandability

Expandability refers to the maximum number of switch ports that can be supported by a particular switch architecture [2]. As stated previously, a time-division switch is limited by the maximum speed of the shared communication path, but the use of stages within a

time-division switch can increase the maximum number of ports at the cost of increased switching delay, and more complex routing algorithms [13]. A non-blocking space-division switch is theoretically infinitely expandable, but realistically limited in the number of interconnections and fabrication complexity [10].

3.2.5 Support for Multicasting

Multicasting requires the delivery of copies of the same cell to several ports. In an ATM network, a multicast connection is established in several steps requiring the establishment of the first connection followed by several add party messages [8]. If the same identifiers can be allocated to multicast cells going out on different output ports, the multicast process is simplified. This method, however, divides the total number of indentifiers among the ports, as an add party message may be made by any of the connections. If the identifier in not the same for each of the output ports, additional table lookup will be required at the output port to generate the correct header for that port.

A time-division switch is inherently better at implementing multicasting as the shared communication path is connected to all output ports. With a time-division switch, appropriate control signals can forward the cell to all the required output ports and indicate if a secondary table lookup is required. A space-division switch on the other hand requires more complex multicasting algorithms. For this switch type, a replicating network can be used to generate several copies of the cell and send them through the interconnection network. This replication process can be performed at the input port, creating the requirement for input buffers capable of storing several cells that are to be routed, or within the SE network as needed to reach the correct output ports.

3.2.6 Implementation Complexity

Implementation complexity for a switching fabric can be expressed in terms of hardware requirements such as gate count and pin-out. As a result, it is hard to compare without the specifications for an actual implementation, though comparisons of several of the architectures have been made based on the chip count by limiting the design to pin-limited chips [14].

3.3 ATM Switch Fabrics

3.3.1 Time-Division Switches

Shared Media

Perhaps the simplest of the ATM switches is the shared media bus or ring architecture as shown in Figure 3.3 [10]. The control logic for such an architecture is an easily implemented TDM arbitration scheme. As all ports are capable of receiving all cells, multicasting is also easily implemented in this switch type, as is priority service. Provided that the shared media can serve all output ports within a switching cycle, output buffering can be used, giving the switch excellent throughput/delay characteristics. The limits of shared media switches lies in the speed of the medium [10]. The media can only operate at a rate limited by the delay of the bus and associated control logic.

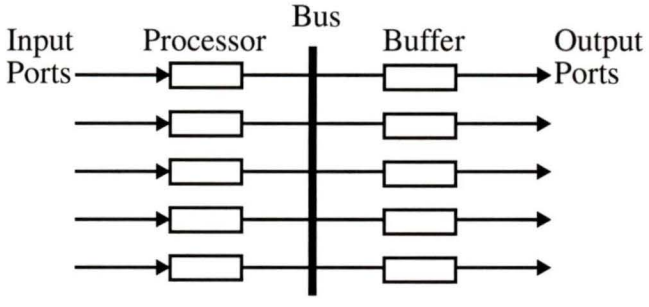


Figure 3.3 Shared media switch.

One recently presented shared media switch is the Bus Structure Switching (BSS) element [15]. This architecture is based on the TDM bus, using a full ATM cell bus width to transfer a cell within one clock cycle. A shift register input scheduler sequentially scans the input ports, and places the results on the 425-bit bus (424-bit cell + 1-bit "Presence Information") on the rising clock edge. This cell remains on the bus until the subsequent rising edge, where the next cell is loaded on the bus. Output buffering is used to solve output port contention. This particular implementation can support 90 ports each with a physical rate of 155 Mb/s [15].

Shared Memory

A shared memory switch, depicted in Figure 3.4, utilizes a high-speed memory shared among a set of input ports to achieve the high throughput/delay characteristics of the

shared buffering scheme [13]. The memory itself must be capable of both reading from and writing to all the ports that it supports within one switching cycle. Therefore the limit on this switching fabric is the memory access time, though this can be alleviated through multiple port memories [13, 27], and parallelization of the cell. Output Scheduling is performed through the use of some sort of ordered storing of memory addresses per output port or cell priority class. Again, multicasting is easily implemented through writing the same memory address to several queues.

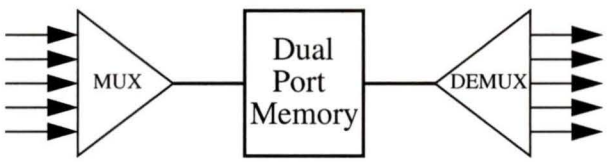


Figure 3.4 Shared memory switch.

In [13], several proposed shared memory architectures are reviewed. The main differences in these approaches lies in the multiplexing technique, and queue maintenance method. In general, the multiplexer schedules memory writes into idle memory addresses stored in an idle address buffer. The memory address is stored in a queue for a particular output port/priority class. The output queues can be stored in either a FIFO of memory addresses, or in a linked-list style. To alleviate port restrictions due to memory speeds, multistage shared memory switches, depicted in Figure 3.5, can be used. The internal link rate between memory stages can be increased, or multiple links between stages can be added to reduce the probability of blocking in multiple stage shared memory switches.

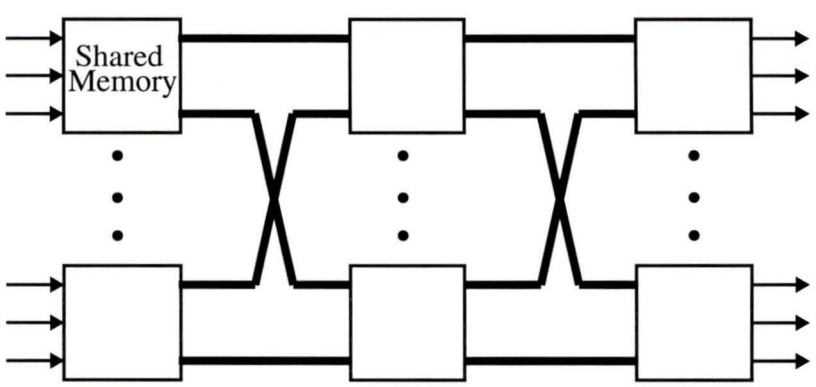
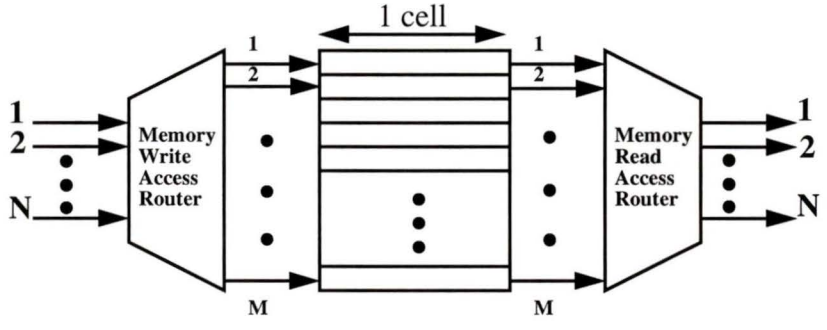
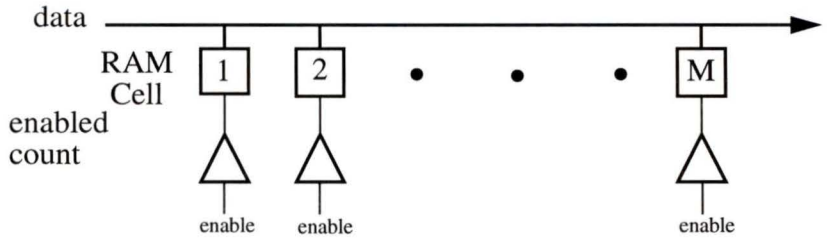


Figure 3.5 Multiple stage shared memory switch.

As ATM switching has developed and experience has been gained with the first generation of ATM switch architectures, new switches have been developed that attempt to improve on perceived difficulties in older architectures and use implementations that reduce the complexity. The PRIZMA switch [16] is a shared memory switch that uses a different queuing technique in order to slow the necessary memory access speeds. Each output queue logically acts as an N input, one output queue with each input port having contention-free access to the queue with a link rate equal to the port rate. Physically, all output queues share one memory. To accomplish the reduction in necessary memory access time, the cells are written as if the memory were composed of many logical shift registers each 1 cell size in length, as shown in Figure 3.6(a). The memory itself is not a physical implementation of a shift register however, but a RAM memory 1-bit wide and 1 cell deep, or a RAM M-cells wide with bit-write capability [16]. Memory addressing in the pseudo-shift register (Figure 3.6(b)) is accomplished through a counter synchronized to the data clock. The 16x16 single chip switch element of [16] can support 800Mb/s per port for an aggregate throughput of 12Gb/s [16]. Beyond 16 ports it is proposed to use several primary elements to build a one-state switch with arbitration between the elements, or a multiple stage switch with interconnected switching elements.



a) The PRIZMA switch.[16]



b) RAM pseudo-shift register

Figure 3.6 The PRIZMA switch [16] memory scheme. (a) The PRIZMA switch (b) RAM pseudo-shift register.

3.3.2 Space-Division Switches

Crossbar

The classical crossbar switch of Figure 3.7 [10, 17] was developed originally for the circuit-switched analog telephone network. The basic structure was a crosspoint architecture that could establish a non-blocking connection between any input port and any output port. The complexity of the crossbar increases as N^2 [10], where N is the number of ports. In general a crossbar switch can be defined as any switch that grows with a complexity of N^2 [10]. As there is only one path between any input and output port, there is a possibility of output port contention in this architecture. For a crossbar operating at the port speed, input or internal buffering must be used to resolve this contention. Input buffering requires external control logic for the crossbar, while internal buffering requires cell buffers at the crosspoints [17]. Output buffering can be used if the crossbar network operates at a multiple of the input port rate.

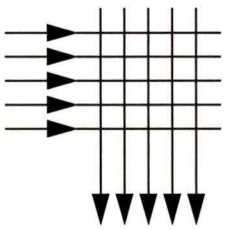


Figure 3.7 The crossbar switch.

The Knockout Switch

A fully-interconnected space-division switch is one in which every input port has a contention free path to every output port. This sort of switch allows output buffering to be used to its advantage and a fabric that operates at the port rate, but requires multiple concurrent memory write operations at the output port.

One switch that implements such an architecture is the Knockout Switch [9]. In this switch, the cells at each of the N input ports are placed on separate broadcast buses. However, the output buffer is not capable of accepting N cells in one cycle. Instead an N-to-L concentrator is used at the output port, as shown in Figure 3.8 [9]. If more than L packets are routed to the same output port within a cycle, the excess cells are dropped. The decision on which cells to drop is based on a competition. The knockout competition chooses

one of two inputs entering a two-port switching element. The winner goes on to the next stage while the loser is eliminated or competes again depending upon the implementation. Any retransmissions of dropped cells are assumed to be done from the endpoints, and not within the switch itself. L can be calculated based on the required cell loss, and can be assumed independent of load for large N [9]. Results indicate that a concentrator of 12 outputs will have a cell loss probability $< 10^{-10}$ independent of the number of ports [9].

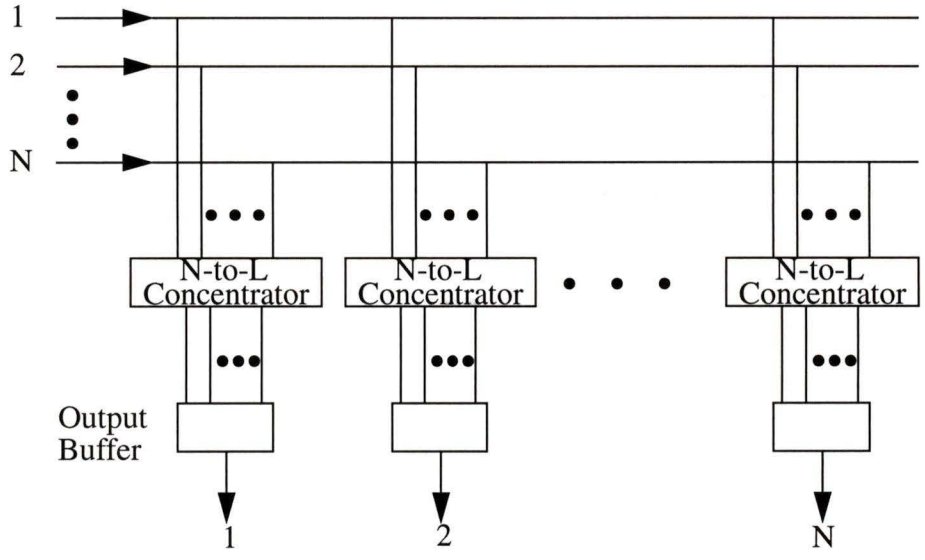


Figure 3.8 The Knockout switch[9].

Multistage Interconnection Networks

The multistage interconnection network (MIN), shown in Figure 3.9, is a common space-division switching technique [17]. The MIN itself is composed of basic switching blocks that handle a subset of the switch ports. Several stages are used to create one or more paths between an input/output pair. A MIN can be blocking or non-blocking, which is governed by the number of paths through the network and any controls/resources in addition to the switching elements. Methods to avoid internal blocking include buffers within the basic switching elements, increasing the internal link speed, using backpressure to delay the transfer of blocked cells, providing multiple links between two switching elements, and providing multiple planes/paths between input/output pairs.

The banyan switch [17], is an example of a multistage interconnection network

with one path between any input-output pair. The complexity of a banyan-style switch is on the order of (N log N) as opposed to the (N^2) complexity of the crossbar switch making it more suitable in terms of complexity for large scale switch [10]. The problem with the banyan switch is that it is blocking, and its performance degrades with size [10].

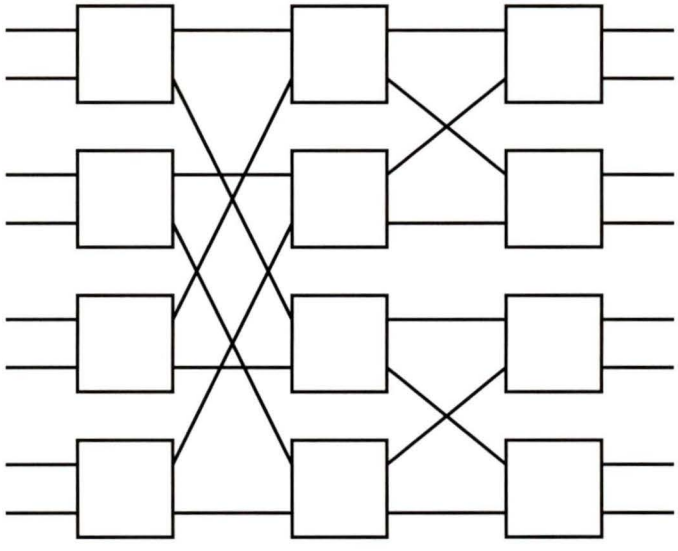


Figure 3.9 Multistage interconnection network.

The delta network [10] is a subset of the banyan type that uses self-routing to traverse the switch. The delta-b network in particular is a common switch architecture consisting of identical bxb switching elements with k stages, where ($N = b^k$). The routing through the network is performed through the analysis of an appended routing tag that is interpreted stage-by-stage.

Using the methods discussed previously to improve the performance of an MIN, there have been several implementations of improved banyan architectures. The batcher-banyan switch is one such example. The batcher-banyan architecture requires a batcher network in addition to the regular banyan network [10, 17]. The batcher network sorts the cells according to their routing tag. After the sorting, the banyan network becomes non-blocking, but there is still the possibility of output port contention. The complexity of the batch-banyan network grows as (N log N^2) compared to the (N log N) complexity of the banyan switch. The buffered banyan switch [17] architecture is another example of an improvement on the basic architecture. The buffered banyan network uses cell buffers

at intermediate nodes to store blocked cells, thus reducing cell loss. An augmented banyan switch [10] is also an architecture building upon the basic banyan network. In the augmented banyan architecture, extra stages are added to increase the number of paths between an input/output pair. The number of paths is generally doubled for every stage added beyond the basic banyan network. The extra stages can not only be used to provide multiple paths, but also to evenly distribute the traffic and improve the performance of the network.

Other methods of increasing the performance of MIN switches include providing several switch planes in parallel to increase both the available paths and the redundancy, and the use of misrouting in the main network with a recirculating network that allows misrouted cells to be sent through the network again [10]. All of these methods increase the control complexity of the basic MIN as decisions must now be made on the path to be taken through the network.

As with shared memory switches, by using the results gathered from early space division switches newer switching fabrics have been developed that improve on the performance of the older generation switches. The Multinet switch [18], depicted in Figure 3.10, is a MIN-style switch utilizing internal FIFO queuing to eliminate contention within the fabric in a buffered banyan style [17]. In this way, it can be seen to be similar to the multistage PRIZMA switch [16], however the SE themselves are not identical across all stages, as shown in Figure 3.10. The buffering scheme in the Multinet switch utilizes partially shared queuing that shares a particular buffer among a set of ports. This is to ensure that a particularly loaded port does not limit the buffer space of all ports, only a small set of them. The Multinet switch maintains the self-routing property found in the delta MIN by using a 1x2 demultiplexer as the basic switching element. The demultiplexer of the i^{th} stage operates on the i^{th} bit of the appended routing tag where there are log *N* stages for an *NxN* switch. The FIFO at each stage may receive up to 2^{n-i+1} cells and transmit 2^{n-i} cells in one switching cycle, where n is the total number of stages. From Figure 3.10, it is evident that there are several paths between an input/output pair, reducing the probability of contention. Therefore the Multinet switch can be classified as a combination buffer/augmented banyan switch. Multicasting in this switch can be achieved be transmitting on both the upper and lower links of the 1x2 demux, and this requirement can be indicated in the routing tag.

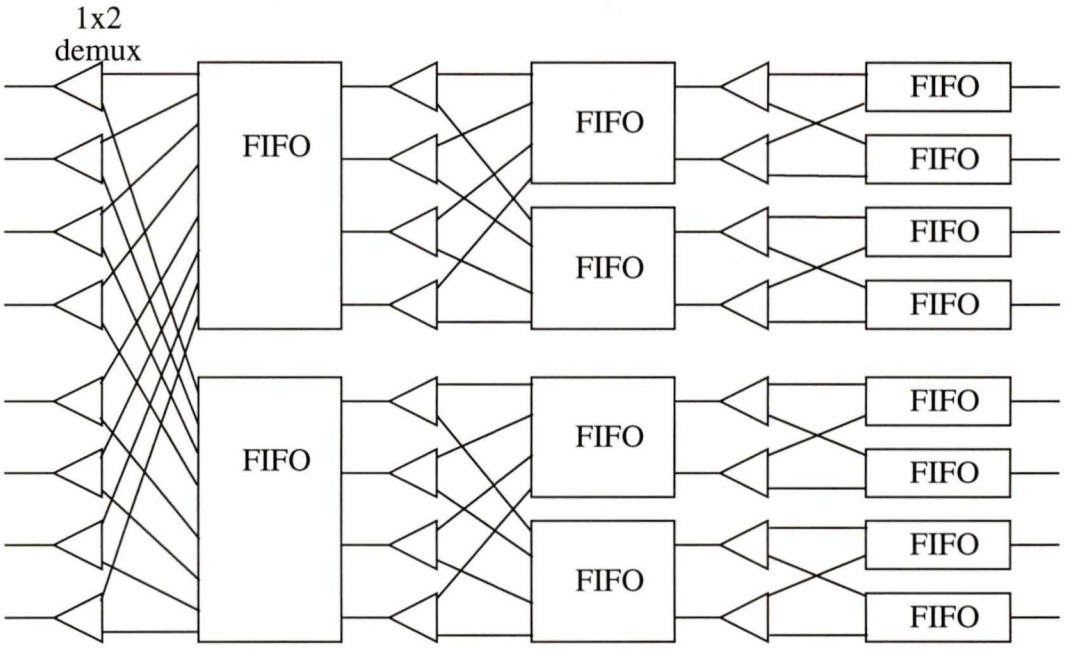


Figure 3.10 The Multinet switch [18].

Another switch improving upon earlier designs is the Tera switch [19], shown in Figure 3.11, which is a hybrid input/output switch utilizing several banyan networks. In the Tera switch, input cells are routed through a non-blocking Banyan network to a completely shared input buffer implementing a FIFO queue. Cells from the top of the queue are fed to a second banyan network that performs cell copying to implement multicasting. An output network is used to route cells from the copy banyan network to the discrete port output buffers that can accept multiple cells per switching cycle. At the input of the output network, a decision is made as to whether the number of cells for one port exceed the number that can be written (over-allocation). If this is found to be true, the excess cells are fed into the third banyan network which operates as a recirculation network. In [19], the Tera switch was simulated for 64 ports and a total of 40 buffers per port (20 dedicated output buffers per port). The results indicated that a doubling of the rate within the output network provided throughput performance comparable to output buffer switches while generally reducing the required internal speed of the output buffered switch.

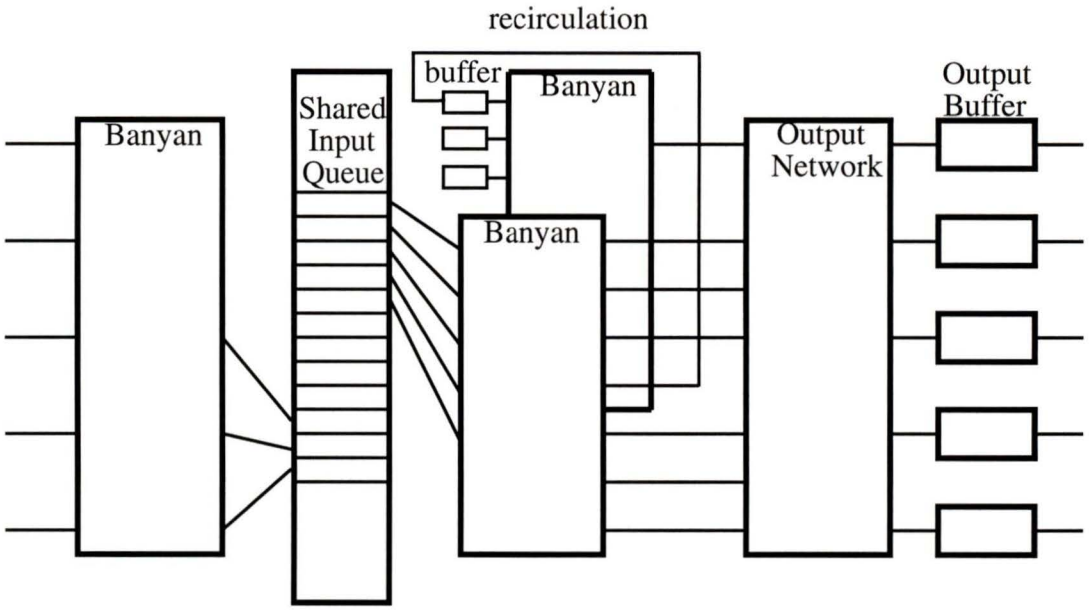


Figure 3.11 The Tera switch [19].

3.4 Switch Comparison

As is evident from the previous switch architecture descriptions, there are a myriad of choices for ATM switching fabrics, each with its own advantages and disadvantages. The main influence on the switching fabric is the expected throughput, as the switching fabric decision should be based on the size of the switch required. For a general division in switch size, there can be a separation of central office (CO) and campus (CA) switches where CA switches generally have throughput < 5Gbps with any switch above this being classified as a CO switch [2]. The actual division line may rise as advance in switching techniques are made and technology improves.

For CA switches, time-division switches are simpler to implement and require less fabrication area than space-division switches [10]. A TDM bus requires only an arbiter and bus while a shared memory switch, if the access time requirements are low, can use a compact DRAM structure. The shared media however is limited in large scale switches as the number of ports competing for an increasingly longer bus rises, or the speed of individual ports increases. For shared memory switches, access times are currently on the order of nanoseconds, and increasing parallelization also increases the ground bounce caused by multiple transistor switchings, reducing some of the gains made [27]. Space-

division switches on the other hand scale well with increasing throughput/port number, though the complexity increases as does the size. Integration on one chip therefore becomes less of a possibility which loses the advantages that go with single-chip design [24, 25]. For small switches, the difficulty of implementing multicasting and the advantages of the time-division switch in complexity and ease of implementation make time-division switches more attractive.

The PRIZMA switch [16] in particular seems like an attractive choice as its memory allocation scheme makes it more scalable than other shared memory offerings. The use of shift register style memory reduces the necessary access time while giving all the advantages of the shared memory switch including simple multicasting and priority queueing. As the RAM only acts as a pseudo-shift register, the individual port rate is limited to the time for one memory write cycle.

3.5 Summary

A brief overview of some of the issues in broadband packet switching has been presented. Several ATM switching architectures have also been outlined, and their advantages/disadvantages have been discussed. From this discussion, a recommendation has been made as to the choice of switching fabric for different switch sizes.

Chapter 4

Switch Overview

4.1 Introduction

The design process described in Chapter 2 outlines the steps that were used to develop the framework of an ATM Layer switch in VHDL. This implementation performs many of the requirements outlined in the ATM standards documents, and also provides a hardware structure capable of supporting the software-based functions of a full ATM switch.

4.2 Basis for Implementation

The main document used for implementation requirements was the ATM Forum UNI 3.1 [8] which outlines the necessary functions for switches supporting a UNI. As such, it is intended for interfaces between ATM user devices and private ATM network equipment, ATM user devices and public ATM network equipment, and private ATM network equipment and public ATM network equipment. It does not cover the interface between network equipment within a network, or between two public networks. Here a private network is defined as one that operates as part of some corporate/campus network and a public network is one that is provided by a public communication network service provider [8].

As a result of the scope of this reference material, this high-level implementation was designed as a local campus area switch supporting a small number of ports, though the input and output processors are modular and could function as part of a larger switch. The UNI 3.1 [8] outlines the required and optional implementation requirements of an ATM switch/device supporting either a public or private UNI. Many of the functions are currently optional, but those that are not include both switched and permanent virtual connections, ILMI, and both point-to-point and point-to-multipoint VCC [8].

4.3 Implementation Details

In the following sections we explain the functionalities of the different modules in the switch implementation. More detailed information is provided in Chapter 5, and full schematics are provided in Appendix A.

4.3.1 ATM Layer Requirements

In addition to the general requirements of an ATM device supporting a UNI, the UNI 3.1 also outlines the requirements of the ATM Layer [8]. These include:

- switched and permanent virtual connections
- point-to-point and point-to-multipoint VCC
- the Interim Local Management Interface (ILMI)
- pass/accept one ATM cell per Physical layer request/indicate
- encode and transmit cells according to the structure and field encoding convention
- multiplexing among different ATM connections
- cell rate decoupling (generating unassigned cells)
- cell-discrimination based on pre-defined header field values
- payload type discrimination
- loss priority indication and selective cell discarding
- traffic shaping
- support of ATM Layer Management Functions
- support of ATM UNI signalling

4.3.2 Implementation Support

The ATM switch of this thesis supports or provides for software-based implementation of the above requirements except for traffic shaping, as this was assumed to be an operation of ATM endpoints [8]. In addition, the implementation includes cell tagging, and a priority service discipline at the output ports.

4.3.3 Switch Clocking and Initialization

In this implementation there is a system clock (**hclk**) with a period equal to the cell transfer time determined by the Physical layer cell transfer capacity, and all data transfer between components is synchronized to this clock to provide a slotted switching mechanism. Below the system clock there is a primitive clock (**pclk**) that transfers the bit serial cell and is also used for the switching fabric. A third clock (**dclk**) is also used for parallel, less rate-dependent data transfer.

For initialization purposes, an initialization signal has been implemented that resets the routing/multicast/traffic tables in addition to resetting all queue pointers. The system clock is also used for resetting counters and flags in several of the modules.

4.3.4 Input Processing

This section outlines the cell processing that occurs at the input port of the switch. For further details refer to Section 5.4.1.

The transmission of a cell from the Physical to ATM layer is assumed to be a bit-serial transfer from the MSB of the header to the LSB of the payload. Cell transfer is indicated by a data flag from the Physical layer. Header processing begins after the first 32 bits have been read and an initial decision as to the cell type is made based on header discrimination. Cells requiring table lookup have their headers forwarded to the routing table. These cells include user and OAM cells, with unassigned, signalling, and ILMI cells not requiring table lookup. A traffic monitoring module lies on the same bus as the routing table and reads user header concurrently with the routing table.

In this work, the table is implemented as a Content Addressable Memory (CAM) algorithm requiring two table searches to differentiate between a Virtual Path switch, a Virtual Channel switch, a multicast cell, or a user-defined ILMI connection. If no entry is found for a header, a table search error is indicated. Any new header is routed back to the header manager along with its status. This new header includes a routing tag for output port address and priority class. Concurrently, the traffic module is performing a compliancy calculation for the cell. The results of this calculation can be a cell tagging, a cell discard, or no action.

The header manager uses a combinational logic array to make the final decision as to the cell type and the outgoing header. This array will also tag the cell low priority using the CLP bit or discard the cell based on the results of the compliance check in the

traffic module. The results of the array are latched on the falling edge of the system clock. The CLP bit of the new header is also inserted as the LSB of the routing tag to be used at the output module.

The body of this cell is concurrently stored in a serial-to-parallel conversion module that discards the old header. This module can store two cell bodies using 16-bit wide, 24-bit long shift register buffers.

On the subsequent **hclk** rising edge, this cell is transferred to one of five modules. These are the user buffer, the signalling buffer, the ILMI handler, the multicast module, and the local management module. The user buffer is a FIFO queue handling - user cells, OAM non-endpoint cells, and any cells generated by the multicast or local management modules - all of which will be routed through the cell switching fabric. The signalling buffer accepts all signalling cells for routing to the signalling module. Both the user and signalling queues are composed of 16-bit wide 27-bit long shift registers, sufficient to store one cell with an appended 8-bit routing tag. The ILMI handler reconstructs the SNMP request and searches its local table for the necessary information. As the resultant message will be sent back across the UNI, the subsequent ILMI cells are sent directly to the output port. The multicast module receives the multicast cells and performs multiple table lookups to retrieve the set of output headers and routing tags. This multicast module can generate up to N cells where N is the number of output ports. These cells are forwarded to the user queue using a bus arbitration scheme controlled by the header manager. Finally, the local management module removes OAM cells that have reached their endpoint, monitors OAM non-endpoint cells passing through the user FIFO, and reads any cells received with header errors. The local management module can also insert cells in the user FIFO again based on bus arbitration controlled by the header manager.

4.3.5 Cell Switching Fabric

This section outlines the switching mechanism within this implemenation. For further details refer to Section 5.4.2.

The current cell switching fabric is a TDM bus implementation, though the outputs of the user FIFO were left generic enough to use the same input module for different switch architectures. Each user FIFO uses one request line to indicate the presence of a cell in the queue. The output of the user buffer is connected to a data bus and a control bus with

a width equal to the routing tag length. The switch controller (in this case a TDM bus arbiter) can grant either data and control, only control, or only data depending on the switching fabric. The TDM bus arbiter as implemented grants both, and resets this grant on the drop of the request line. The bus arbiter polls the request lines in a round-robin fashion on the rising edge of **pclk**. It will only grant the bus if there are enough **pclk** edges left before a negative transition of the **hclk** clock. Currently the bus allows the transfer of 16 cells within one switching cycle, where 1 switching cycle is 1/**hclk**.

4.3.6 Output Processing

This sections outlines the buffering that is performed at the output ports. For further details refer to Section 5.4.5.

Output processing performed at a port deals with scheduling and queuing. The output buffer maintains separate queues for signalling, management, ILMI, and eight classes of user cells based on their QoS requirements. The output buffer must grant any request made by the signalling, management, and ILMI modules to write to their respective output port queues. For signalling and management cells, this is accomplished through a request line in conjunction with a port address in addition to a resolved grant line. The ILMI module will write directly to an output buffer on the same interface card, but still using a request/grant scheme. For switched user FIFO cells, the output buffer has no control over the granting process, as it is handled by the TDM bus arbiter. Therefore, user cells are queued in a separately accessed queue from the signalling, management and ILMI cells to avoid conflict between write operations. Currently, each output port has 64 cell buffers shared among the 8 priority classes. The shared buffer approach reduces the probability of buffer overflow for a particular class, but can result in one class taking all the buffer space. This can be avoided by limiting the maximum buffer occupancy of a particular class. Additionally, the user buffer will not accept low-priority cells marked by the CLP, indicated by the LSB of the routing tag equal to 1, when the buffer occupancy of the queue passes a programmable threshold. This selective cell discard reduces the probability of cell loss for high priority cells beyond that of only shared queuing.

An output cell buffer itself is a 16-bit wide shift register structure with eight shift registers of length 26 and eight shift registers of length 27. This staggered shift register structure shifts out the appended routing tag to transfer a correctly coded cell to the Physical layer when enabled properly. The buffer is written to in parallel, and is read from serially. The output of a shift register buffer is high impedance unless enabled.

Each of the queues maintains a flag line indicating the presence of a cell. These lines are fed to the scheduler that grants access to the Physical layer interface when a request is made. On a rising edge on the Physical layer request line, the scheduler polls the flags from the queues, granting access to the queue with the current highest priority. A grant to a queue prompts the queue to set the data flag to the Physical layer, and begin a bit-serial transfer of the cell from the MSB of the header to the LSB of the body. If all queues are empty when a request is made, an unassigned cell is passed to the Physical layer.

4.3.7 Signalling and Management Modules

Both the signalling and management modules are software-based processors accessed through a 16-bit data bus with identical bus arbitration methods. In addition to the processor, each has an associated arbiter that controls access to the data bus. The arbiter handles requests from the input ports and generates output port requests initiated by the processor.

From the signalling module, additions may be made to the routing table through an address bus and control/data bus. Using these buses, new entries can be added, in a bit-serial fashion, to the traffic table or the routing/multicast table, and old entries can be removed. The signalling module must also generate the internal routing tag for any new connections. This includes assigning one of eight priority classes to user cells based on the information in the source traffic descriptor.

4.3.8 Modularity

The VHDL implementation of this design is highly modular, providing for a regular design and easily implementable design changes. The functions of the input and output processors are replicated for each port. The signalling and management modules are independent of the number of ports, and are separate from the bus arbitration process. In addition, the bus arbiters and buffer controllers are common with the exception of the user output buffer, which does not grant requests. The buffers themselves are also regular and modular, based on a small set of common registers. As the output scheduler only depends on the indication of queue occupancy it is independent of the physical implementation of the output buffers, as the output buffers are independent on the priority scheduling algorithm implemented.

4.4 Performance

The performance of the switch in terms of throughput/delay will be that of a hybrid input/output buffer time-division switch. Congestion and buffer overflow will result at the input ports under a heavy multicast load, as the switching fabric can currently only handle 16 cells per switching cycle. This can be alleviated by using a TDM bus clock faster then the input port rate to service multiple cells per input queue per cycle. At the output port, as all output buffers are shared among all priority classes, the CLR will be that of an N-port switch with 64 output buffers per port. This buffering scheme gives a cell loss probability $< 10^{-12}$ for loads having $p<0.8$ [11].

More important is the delay of input processing and controlling the flow of signalling and management cells so as not to inhibit user data cells. By separating the cell header for preprocessing and using concurrent table search and traffic monitoring, the input processing can be performed in one switching cycle. In an uncongested switch, the cells traverse the complete switch in three switching cycles from the reception of the MSB of the header from the Physical layer to the transmission of the LSB of the body to the Physical layer. Any additional switching delay is a result of Physical layer processing. At the SONET rate of 155.52 Mb/s, this would correspond to a delay of $< 10\mu s$, which is comparable to other switches [2].

4.5 Summary

In this chapter, an overview of the VHDL implementation of the ATM switch described in the Chapter 5 has been given. This implementation provides many of the requirements of an ATM Layer switch, as well as providing a framework for implementing software-based solutions to other aspects of a full ATM switching solution. In the next chapter this implementation is described in more detail with a breakdown into low-level components and the provision of timing diagrams for cell flow across the switch.

Chapter 5

VHDL Design of an ATM Switch

5.1 Introduction

In Chapter 2, the basic elements of an ATM switch were presented in a generic fashion. In Chapter 4, an overview of a specific high-level VHDL design of an ATM switch was provided. In this chapter we discuss this implementation in further detail. Schematics for certain key components are provided, as are timing diagrams for data transfer and control within the switch. Full schematics with a component list and port information can be found in Appendix A. For further details about the VHDL code, refer to Technical Report VMC/2 [29].

5.2 Very Brief Overview of VHDL

One of the major advantages of using VHDL [26] is its ability to describe a system at a much higher level of abstraction than the gate level. Using this ability, the logic and algorithms used for system data flow can be verified before a large amount of effort has been made to design a fully synthesizable gate-level design. This high-level design and modelling can be simplified using many behavioural language constructs similar to those found in other high-level programming languages. The hardware-based nature of VHDL allows the design to be easily divided into small, modular components.

5.2.1 Basic Elements

The basic unit of VHDL is the ENTITY that describes a device, its interfaces, and any other device attributes. The function of an ENTITY is defined in an ARCHITECTURE that specifies the logic of the ENTITY. This logic can be a behavioural, structural, or data-flow description. A *behavioural* ARCHITECTURE uses language constructs like

loops, if statements, case statements and other high level routines that form an algorithmic description of performance. A *structural* model uses interconnected COMPONENTS in a device-level model. These lower-level COMPONENTs are also described by an ENTITY and ARCHITECTURE, and are declared as a COMPONENT in the higher layer ARCHITECTURE declaration region. The *data-flow* description is similar to a *Register Transfer Language* (RTL) description.

5.2.2 Hierarchical Design

To implement a structured VLSI design, a layered structural description can be used to break the design into small COMPONENTs. For the ATM switch, the highest layer defines the input and output ports and any additional ports that are used for switch control/management. Above this is the PACKAGE that defines data types, constants and subroutines that will be used throughout the design. The constructs within this PACKAGE can be accessed using the LIBRARY command, and the USE statement. Below this top layer, the switch functionality is broken into COMPONENTs. The number of COMPONENTs at a sublayer should be limited so as to make the schematic of the layer relatively simple to follow. This process of designing structural layers with COMPONENTs and port mappings should continue until each COMPONENT performs a small set of functions. At this low level, the performance can be described in either a concurrent or sequential fashion using behavioural descriptions if high-level modelling is used. The basis of concurrent description is the BLOCK statement that groups concurrent statements that perform a set of functions. The BLOCK statement however is not necessary to implement concurrent statements, but it does provide a means to use clocked events and bi-directional multiply-driven buses. The PROCESS statement is used to group sequential statements, and must be present if sequential statements are to be used. The PROCESS statement itself is a concurrent statement, but the operations within a PROCESS are executed sequentially.

5.2.3 Clocking

To use clocked logic, BLOCKs and PROCESSes can be made sensitive to events on the clock signal. As a signal has both a current value and associated ATTRIBUTEs including its last value, and whether an event has occurred on the signal. These ATTRIBUTEs can be used to define both rising and falling edges of the clock. The BLOCK can be made sensitive to these events using the GUARD statement and a guarded signal assignment

that only schedules an event on a signal if the boolean GUARD statement is true. The PROCESS statement uses either a sensitivity list and IF THEN statement, or a WAIT statement. A sensitivity list declares the signals a process is sensitive to, and invokes the PROCESS on a change in any of these signals. This sensitivity list can then be used with the IF THEN statement to define the rising edge. The WAIT statement defines a boolean expression that suspends the operation of the PROCESS until the boolean is true. Other asynchronous aspects of the logic such as preset and preclear can also be defined using these methods.

5.2.4 Delays

When behavioural modelling is used, there are no inherent device delays in device libraries. However, signal rise and fall times can be expressed using the AFTER clause in a signal assignment statement. If several instances of the same COMPONENT are used, each driving different capacitances, different delays can be specified using a GENERIC statement in the ENTITY definition and a GENERIC MAP in the ARCHITECTURE body. If no delay is defined, then a *delta delay* is used when simulation is carried out. This *delta delay* is assumed to be some value greater than 0, but less than one timestep of the simulator. Therefore a signal assignment statement schedules an event on the signal that occurs a minimum of 1 delta delay after the assignment.

5.2.5 Variables and Signals

A SIGNAL assignment statement within a VHDL design is not updated until the next simulation cycle. To update values within a PROCESS, VARIABLES can be used. These VARIABLES, which can only be used inside a PROCESS, are updated immediately upon assignment.

5.3 Timing, Buses and Signalling

5.3.1 Switch Clocking and Latency

A Physical layer protocol carrying ATM traffic provides an ATM cell transfer rate capacity equal to the line rate minus any bandwidth required for Physical layer overhead. This cell transfer capacity in bits/s is used as the basis for timing throughout this ATM switch implementation. The time to transfer one ATM cell at this rate is used as a high level clock that controls the flow of cells across the switch. The latency of a cell at a stage of

the switch can be no longer than this high level clock period, denoted as **hclk**. It is assumed that each new **hclk** can bring a new cell from the Physical Layer, or require a cell for the Physical Layer. It is also assumed that the Physical Layer is synchronized to **hclk** to provide for cell transfer within one positive **hclk** period. It is through using this definition of **hclk** that the switch latency can be expressed as a function of the number of switching cycles a cell is delayed within the switch.

In addition to **hclk**, one primitive clock, denoted **pclk**, is used to transfer the actual cell. Serially, there must be 424 **pclk** periods (8x53) within one **hclk** period to transfer a cell. The actual number used in this implementation is 512 **pclk** periods per **hclk** period, with 480 **pclk** periods while **hclk** is high. In addition to serial cell transfer, **pclk** is used for shared medium switching fabrics where the switching fabric bandwidth is equal to the sum of the port bandwidths and parallel cell transfer is used to speed up cell transfer.

A third clock, the data clock (denoted as **dclk**), is used to transfer information across the switch, and can additionally be used to transfer cells for time-division switches that operate at the port speed, thus allowing the fabric speed to be reduced.

These clock periods can be calculated, assuming that the ports operate at the SONET rate of 155 Mbps, as follows. For this Physical layer, the cell transfer capacity is 149.76 Mbps [8]. To transfer one 424-bit cell at this rate requires approximately 2.83 μs, which is the period of **hclk**. For 512 **pclk** periods within one **hclk** period, a 5.5 ns **pclk** period is necessary. The period of **dclk** is set at 4 times that of **pclk**. Figure 5.1 shows the timing relations of the three clocks: **hclk**, **dclk** and **pclk**.

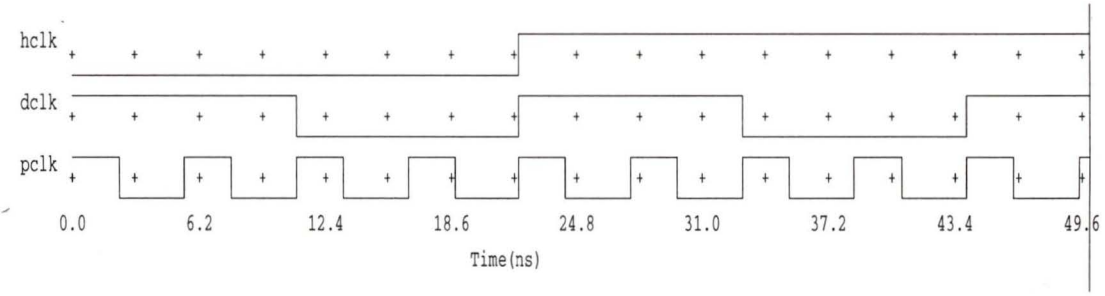


Figure 5.1 ATM switch clocks.

5.3.2 Buses and Interconnections

Though the input and output ports of the ATM Layer switch are serial, within the switch a parallel 16-bit bus is used to transfer cells. Assuming the insertion of an 8-bit header for routing/priority, it requires 27 clock pulses to transfer an ATM cell. Therefore, with some overhead for control, 16 ATM cells can be transferred within one **hclk** period, defining the maximum number of ports for a shared medium switch. Any increase in port number beyond this would require a widening of the bus, or a faster cell transfer clock, which is **pclk** in this implementation.

The data buses in this implementation include one signalling bus, one management bus, and one user bus for each input port. Depending on the switching fabric, the user buses may be separate buses, or one bus that branches into each **SIM** and **SOM**, as is done with this TDM bus architecture.

Within this implementation data transfer between components is performed synchronously. Input data is sampled on the rising clock edge. Output data is placed on the bus with any data flags on the falling edge. Request flags are set on the falling edge, with any grant signals being set on the rising edge. This well-defined clocking pattern makes interconnections and the use of standard cells simple.

5.3.3 Bus Resolution and Propagation

In this implementation four logic states were used, as shown in Table 5.1. At simulation time zero, all signals were initialized to the unknown state. VHDL does not allow a signal to be multiply driven unless a resolution function is used. For the multiply driven, bi-directional buses of this switch, a wired-x resolution function was used. The wired-x function returns an unknown value on a bus if two or more drivers of a different logic value other than high impedance are attempting to drive the bus. The results of a wired-x resolution are shown in Table 5.2.

State	Type
Z	high impedance
0	logic low
1	logic high
X	unknown

Table 5.1. Four state logic.

	X	0	1	Z
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
Z	X	0	1	Z

Table 5.2. Wired-x resolution.

The results of this function call are found by examining the drivers pair-wise, and returning the table value found at the intersection of the two drivers.

5.4 ATM Layer Switch

This section describes the hierarchy of this ATM switch implementation. Further details on each of the components and the functions of their ports can be found in Appendix A.

The top layer of this design is the component **ATM Layer Switch** as shown in Figure 5.2 for an 8-port switch. The pins to this components include the clock signals, an initialization signal, and a set of 5 pins per Physical layer interface (switch port). Each port requires **indicate** and **data_in** signals for cells from the Physical layer, and **request**, **dout_flag**, and **data_out** signals for transferring cells to the Physical layer.

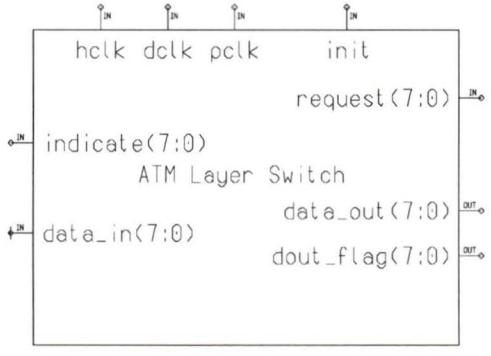


Figure 5.2 Icon of ATM layer switch.

Below this top layer lies the structural architecture of the switch, shown in Figure 5.3. The switch has been decomposed into five main components: **Input Module**, **Connection Admission Control**, **Cell Switching Fabric**, **System Management**, and **Output Module**. This schematic represents the cell flow across the switch more than a synthesized schematic, as the **Output Module** will wrap its

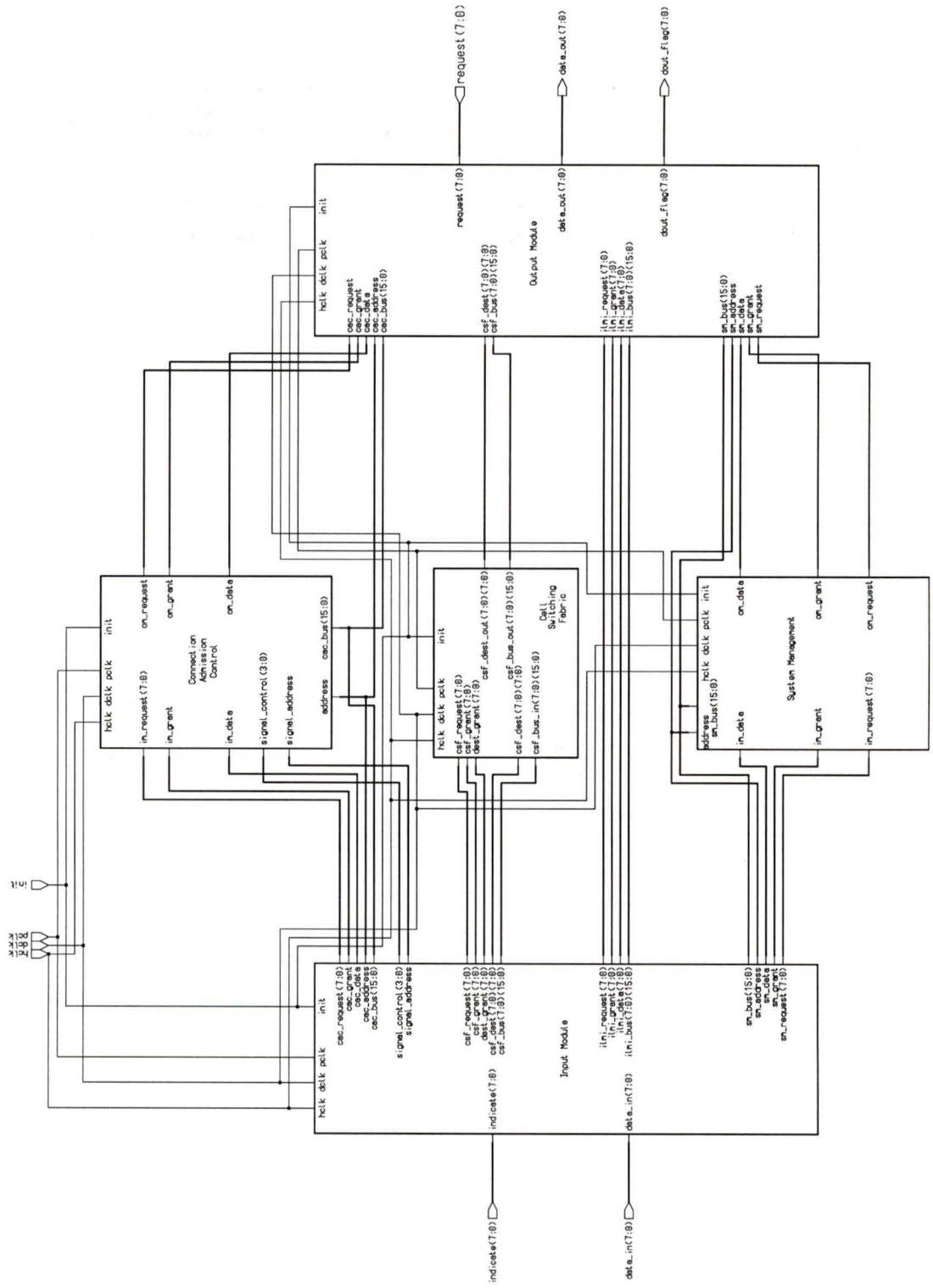


Figure 5.3 ATM layer block diagram.

outputs around to the same interface card from which cells are transferred to the **Input Module**. From the schematic, it is evident that the cells from the **Input Module** are divided into the three planes of the ATM model for handling, before being recombined for output processing. There are individual buses for signalling and management cells, as well as a user data bus per input port. ILMI cells bypass the switching fabric and are transferred directly to the **Output Module**. The **CAC** module also has signal control and signal address buses to make changes to the routing tables in the **Input Module**.

ATM Layer Switch Timing

The timing diagram of Figure 5.4 shows the transfer of ATM user cells across the switch. For further detail as to how this and subsequent timing diagrams were generated, refer to Appendix B. Within the first **hclk** period, a cell is read from the Physical layer through input port 0. Within this period the header is processed, and a routing decision is made. At the subsequent **hclk** rising edge, the cell is transferred to the **User FIFO** within its **SIM**. When this is done, a request is made of the **CSF**, which is granted, and cell transfer to the appropriate **SOM** is made, which is port 3 in this case. The next request signal sees the transfer of this cell with its updated routing tag to the Physical layer. During the second **hclk** period, there was also a request made of an empty buffer, causing the generation of an unassigned cell in the **SOM**.

5.4.1 Input Module and Sub Input Module

The **Input Module** is composed of a set of **Sub Input Modules** of which one is provided per port. The **SIM** itself is shown in Figure 5.5, with its structural architecture depicted in Figure 5.6. The **SIM** is accessed from the Physical layer through the **data_in** and **indicate** pins. Within the **SIM**, both the **Serpar** and **Cell Sort** modules read an incoming cell. When the header has been read, it is the **Cell Sort** module that makes the routing decision, and accesses the **Route Table** or **Traffic** modules if appropriate. The **Serpar** module concurrently buffers the cell payload. On the subsequent **hclk**, the **Cell Sort** module controls the transfer of assigned cells to one of five modules: **CAC FIFO**, **User FIFO**, **Multicast**, **Local SM**, and **ILMI**. It is these five modules that interface with the other components of the switch. New additions to the table in the **Route Table**, **Traffic**, and **Multicast** modules are made through the **signal_control** and **signal_address** pins. The functions of each of these modules are further described in the following sections.

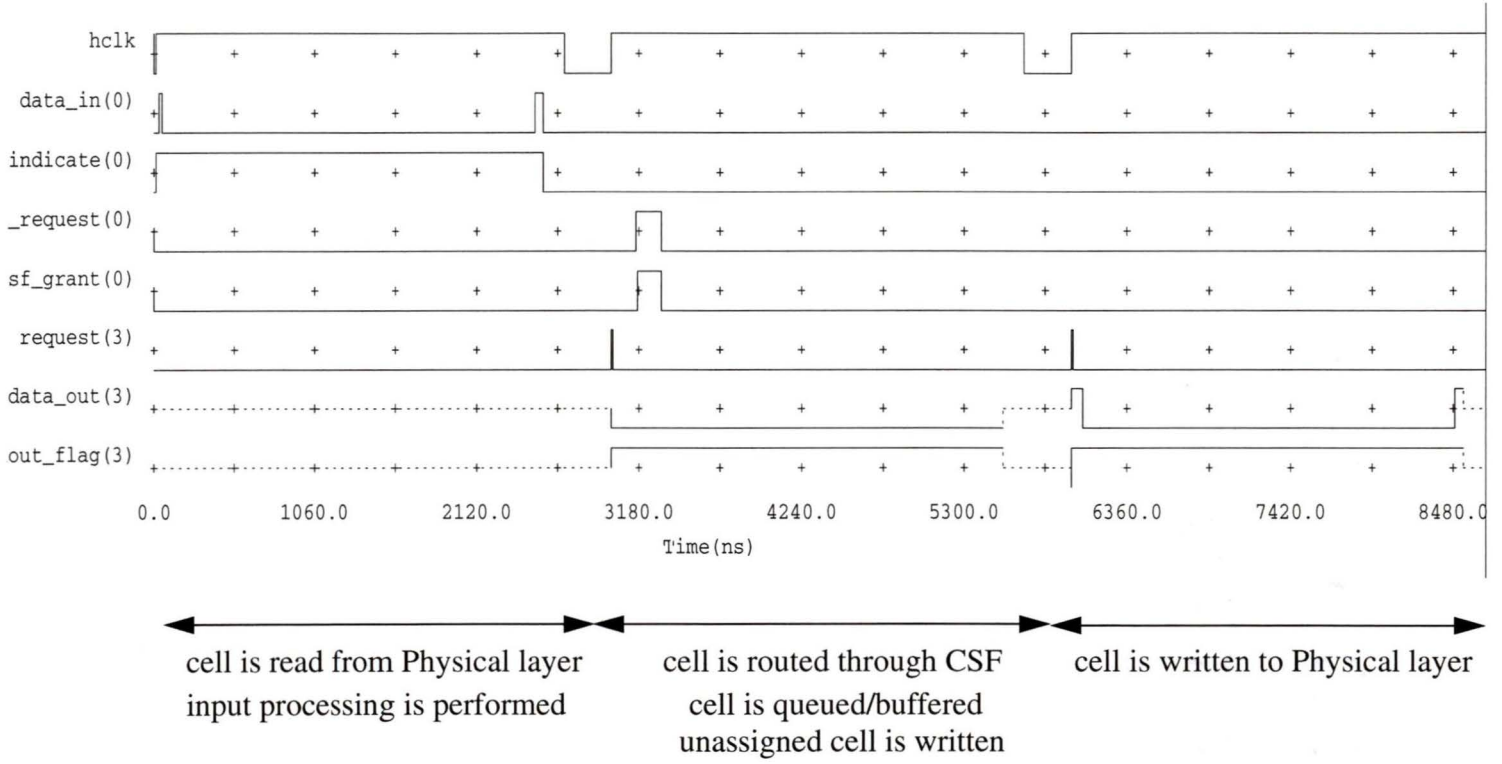


Figure 5.4 ATM layer switch timing diagram.

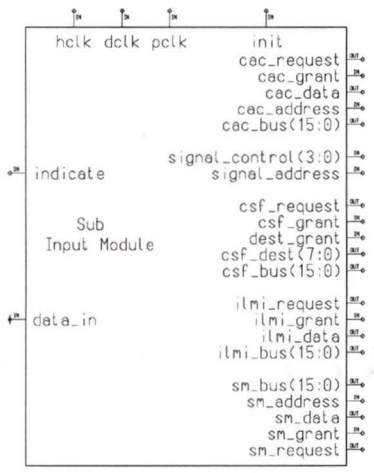


Figure 5.5 Icon of Sub Input Module (SIM).

Sub Input Module Timing

In Figure 5.7, a timing diagram of the **SIM** for a cell requiring table lookup is shown. In this figure, the arrival of a cell is indicated by the rising edge of the **indicate** signal. After the header has been read, a preliminary decision is made as to the cell type. In this case the **Cell Sort** module determines that this cell requires table lookup, and request is made of the **Route Table** using **table_request**. When granted with **table_grant**, the old header is transferred to the **Route Table** module over a 16-bit bidirectional **table_bus** in conjunction with the **table_data** flag. This transfer is accomplished in 2 **dclk** periods. After table lookup is performed, the **table_done** flag is set and the results are placed on the **table_status** bus. The new header, if any, is transferred back to **Cell Sort** on the **table_bus** in 3 **dclk** periods. From this time until the falling edge of **hclk**, the **Cell Sort** module determines the type of cell and the appropriate actions to perform on the subsequent **hclk**. In this timing diagram, the **Cell Sort** module determines that the cell is a user cell, and transfers the cell to the **User FIFO** by setting the **sort_data** flag and the **sort_type**. The new header is transferred in the first three **pclk** periods, after which the **Serpar** module is enabled with the **d_ok** flag and the payload is transferred. After the cell has been queued, the **User FIFO** makes a request for the **CSF** on **csf_request**.

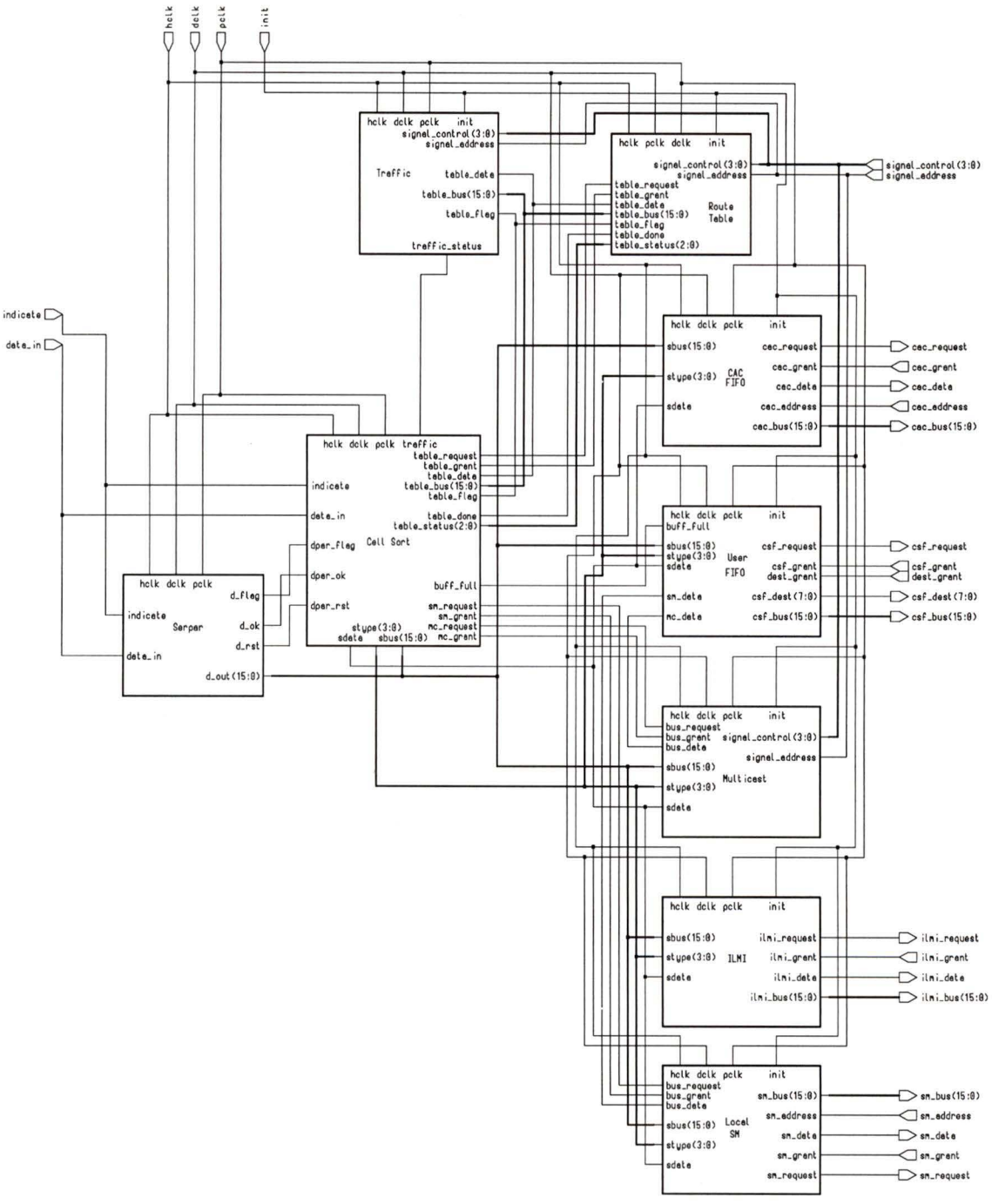


Figure 5.6 Sub input module block diagram.

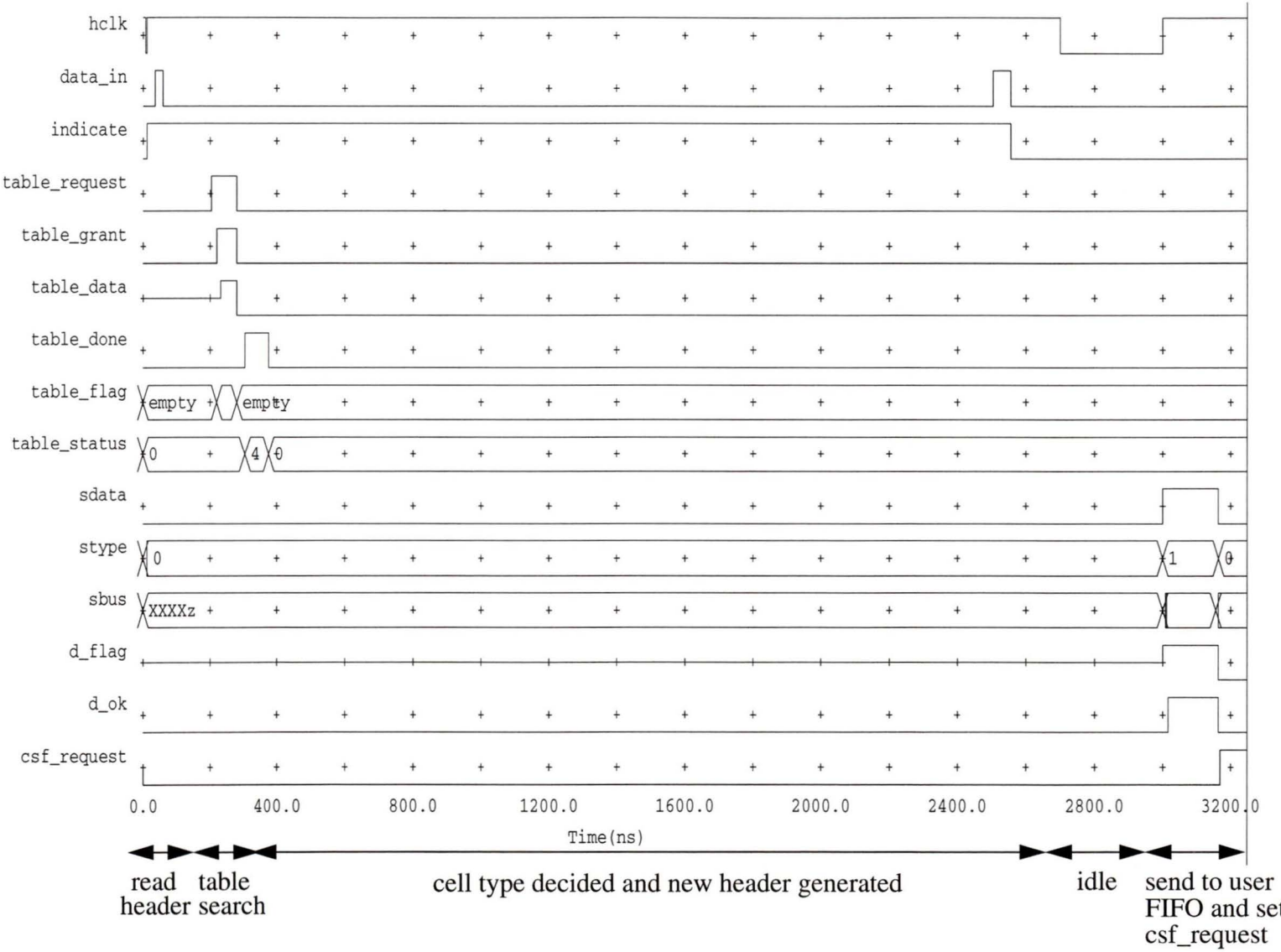


Figure 5.7 Sub input module timing diagram.

5.4.1.1 Serpar Module

The **Serpar** module, shown in Figure 5.8, stores the serial-in input cell payload in one of two 16-bit wide, 27-bit long shift registers. This process allows for simple serial to parallel conversion while buffering the cell during header processing and discarding the old cell header. The shift registers are enabled by the **indicate** signal and a chooser within the module that rotates between the two buffers. Each of the 16 shift registers within a buffer can also be separately enabled to allow for serial-to-parallel conversion using a ring counter. On the subsequent **hclk** period after buffering, the **d_flag** signal is set indicating the presence of a cell if **Serpar** is occupied. Cell transfer onto the 16-bit wide data bus, **d_out**, will commence with the **d_ok** flag, or the flags will be reset and the cell abandoned if the **d_rst** flag is set. The **d_out** bus is driven to high impedance unless the **d_ok** flag has enabled cell payload transfer.

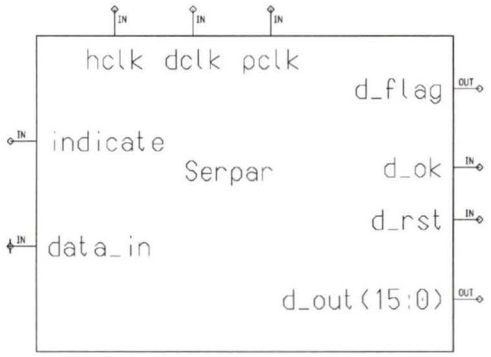


Figure 5.8 Icon of Serpar module.

5.4.1.2 Cell Sort Module

The **Cell Sort** module of Figure 5.9 is the key element of the **SIM** that gathers and processes all information about the incoming cells. When the incoming header has been shifted into a serial-in-parallel-out register, a preliminary decision on the cell type is made based on header discrimination. Based on this decision, one of three actions can be taken, as shown in Table 5.3. An unassigned cell receives no further processing, and the **d_flag** from **Serpar** will be reset on the subsequent cycle. Both signalling and ILMI cell headers are sent to the final decision stage. All other cells are forwarded to the routing table in the **Route Table** module. As the cell type is indicated by the **table_flag**, the **Traffic** module will also concurrently read user headers for UPC processing.

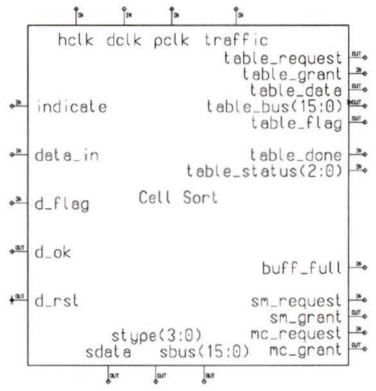


Figure 5.9 Icon of Cell Sort module.

In the final decision stage, a header and cell type are generated for use in the next cycle. This final decision is based on the initial decision and the results of the table search, if any. The essence of this final decision process is the routing of the cells to one of the five **SIM** output components. In Table 5.4, these final results are presented. From this table it is evident that all cell types are forwarded to their respective handlers on the subsequent cycle with the exception of OAM non-endpoint cells which are routed through the **CSF**. Additionally, all errored headers are forwarded to the **Local SM**, and their payloads are discarded.

The final decision is made through an array of combinational logic that outputs the final cell type, from which the header decision is based. Additionally, the results of the **Traffic** module are used to tag the CLP-bit, or discard the cell by redesignating the cell empty. Finally, the CLP-bit is inserted in the LSB of the routing tag. An algorithmic depiction of this decision process is shown in Figure 5.10.

The results of this decision process are latched on the falling edge of **hclk**, and cell transfer is performed on the subsequent rising edge. This is accomplished by setting the **sdata** flag and placing the cell type on the **stype** bus. The first three **pclk** periods are used to transfer the header before the bus is handed over to the **Serpar** module if necessary. After the transfer is complete, the **Cell Sort** module can grant the data bus to either the **Multicast** or **Local SM** modules as required.

cell type	action
unassigned	discard
signalling	final decision
ILMI	final decision
OAM VP, end-to-end/segment	table search
OAM VC, end-to-end/segment	table search
user	table search

Table 5.3. Preliminary cell sort decision.

preliminary	table result	action on next cycle
signalling	N/A	to cac fifo
ILMI	N/A	to ILMI
ALL	error	header to local SM, reset serpar
user	multicast	to multicast
user	other	to user fifo
OAM VP segment	ALL	to local SM
OAM VP end-to-end	VC switch	to local SM
OAM VP end-to-end	VP switch	to user fifo
OAM VC segment	VC switch	to local SM
OAM VC segment	VP switch	to user fifo
OAM VC end-to-end	not error	to user fifo,monitored by local SM

Table 5.4. Final cell sort decision.

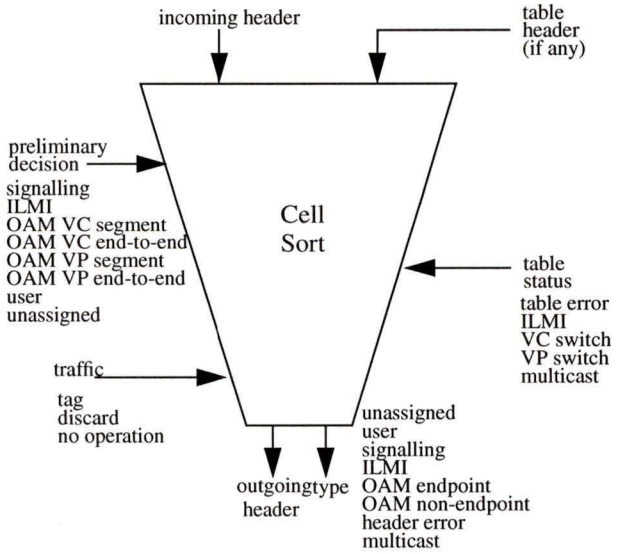


Figure 5.10 Cell sort operation.

5.4.1.3 Traffic Module

The **Traffic** module, shown in Figure 5.11, reads user headers off the **table_bus** as indicated by the **table_flag**. Within this module lies the software-based UPC that enforces the traffic contract. The results of the UPC are sent back to the **Cell Sort** module through **traffic_status**. The possible results of the UPC are no action on the cell (compliant connection), cell tagging, and cell discard. The **traffic_status** is reset to no action on the rising **hclk** edge for the next cell.

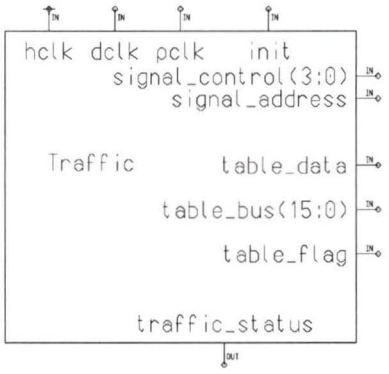


Figure 5.11 Icon of Traffic module.

5.4.1.4 Route Table Module

The **Route Table** module of Figure 5.12 reads user and OAM cells from the **Cell Sort** module and searches its CAM for a new header. The form of a table entry is shown in Figure 5.13. Each entry is 64-bits and contains the incoming path identifiers, outgoing path identifiers, and the routing tag. The table search itself is carried out in two parts using a CAM that indicates the presence of multiple table matches [28].

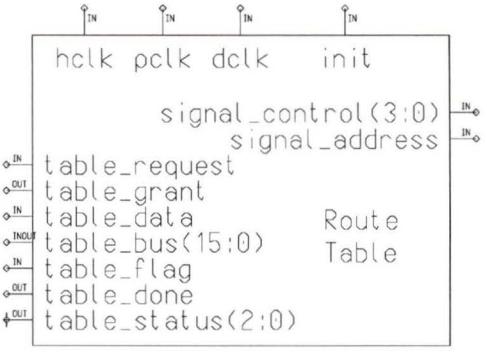


Figure 5.12 Icon of Route Table module.

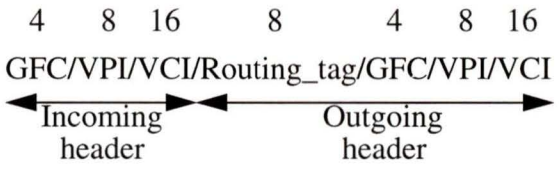


Figure 5.13 Table entry.

The first search is performed using only the GFC and VPI, the possible results of which are summarized in Table 5.5. If no matches are found, a table error message is relayed to the **`Cell Sort`** module. For one match in the first search, the route table module differentiates between a VP switch and a VC switch by examining the incoming VCI entry in the CAM. A zero entry indicates a VP switch, and the old VCI is appended to the outgoing VPI. For a VC switch with one entry, both the outgoing VPI and VCI are used. If more than one entry is found, this can indicate a VP multicast or a VC switch (point-to-point OR multicast). A VP multicast is indicated by multiple entries with zero incoming VCI. If the result does not indicate a VP multicast, a second table search is prompted. There is also the possibility that the search header is a configurable ILMI cell header, which it transmitted to the **`Cell Sort`** module,

Matches	Meaning	Action
0	table error	inform cell sort
1	VP switch OR VC switch (1 channel on path)	send new header to cell sort
>1	VC switch OR multicast	indicate multicast OR prompt second table search

Table 5.5. Possible results of first search.

The second table search, if required, uses the GFC, VPI and VCI of the incoming header. In Table 5.6 the possible results of this search are shown. Again, no matches indicates that a table error has occurred. One match is used to indicate a VC switch, and a new header is sent back to **`Cell Sort`**. If more than one matching entry is found, a multicast signal is sent to cell sort.

Matches	Meaning	Action
0	table error	inform cell sort
1	VC switch	send new header to cell sort
>1	VC multicast	indicate multicast

Table 5.6. Possible results of second search.

5.4.1.5 CAC FIFO Module

The **CAC FIFO** module, Figure 5.14, maintains a FIFO queue for signalling cells. These are cells read through the **sbus** port from the **sbus**. When occupied, the **cac_request** flag is set. The **CAC** grants access to the **cac_bus** through the **cac_grant** and **cac_address**. When **cac_grant** is set and the port address equals the **cac_address**, the **CAC FIFO** module will send a cell to the **CAC** on the **cac_bus** in conjunction with the **cac_data** signal.

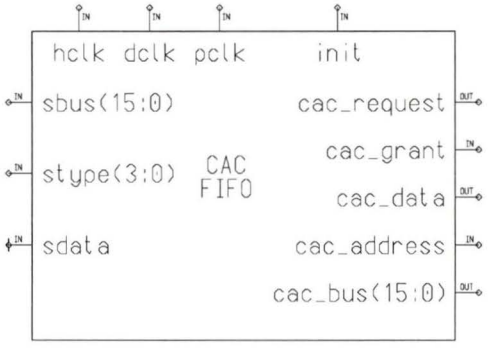


Figure 5.14 Icon of CAC FIFO module.

5.4.1.6 User FIFO Module

The **User FIFO** module, Figure 5.15, maintains a FIFO queue of cells destined for routing through the **CSF**. The queue reads user and OAM non-endpoint cells from the **Cell Sort** module, as well as **Local SM** and **Multicast** cells that have been granted bus access by the **Cell Sort** module.The **User FIFO** module also notifies the **Cell Sort** module when full to avoid lost cells from the **Multicast** and **Local SM** modules. When occupied, the **User FIFO** sets its **csf_request** flag. A **csf_grant** signal

enables the module and it writes the next cell to the **csf_bus**, and places the routing tag on the **csf_dest** bus for the duration of the transfer. When cell transfer is complete, the **csf_request** flag is reset for one **pclk** period to allow the **TDM Bus Arbiter** the opportunity to fairly allocate the resources. When not enabled, the **User FIFO** drives all outputs to the **CSF** high impedance.

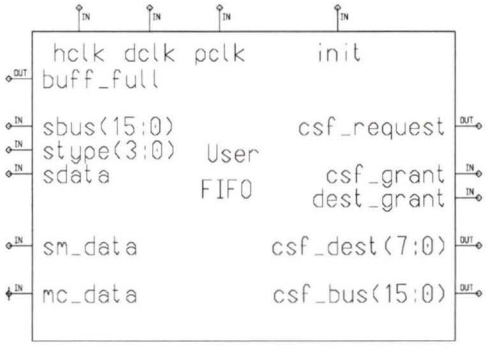


Figure 5.15 Icon of User FIFO module.

5.4.1.7 ILMI

The **ILMI** module, Figure 5.16, handles the ILMI cells sent across the UNI. This software-based module reconstructs the higher-layer message from a set of ATM cells. Table lookup is used to find the answer to the query, and the results are segmented into cells and forwarded to the **SOM** for transmission back across the UNI.

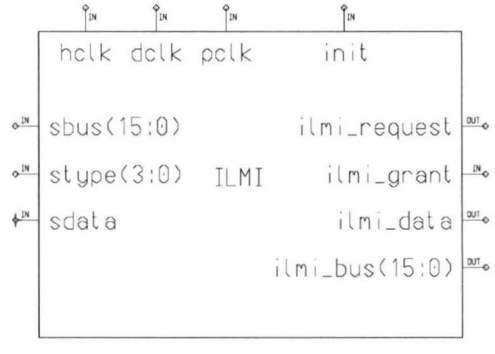


Figure 5.16 Icon of ILMI module.

5.4.1.8 Multicast

The **Multicast** module, Figure 5.17, receives multicast cells as found by the **Route Table** module. The **Multicast** module maintains the same CAM as the **Route**

Table module, but with header buffers equal to the number of output ports. When a multicast cell is received, the table search algorithm is used to recover all outgoing headers. Access to the bus is requested, and when granted by the **Cell Sort** module, a cell is clocked out in conjunction with the **bus_data** flag. This process of request, grant, and send is repeated until all cells are forwarded. The payload of the cell is maintained in a recirculating shift register buffer until the transfer of all cells is completed.

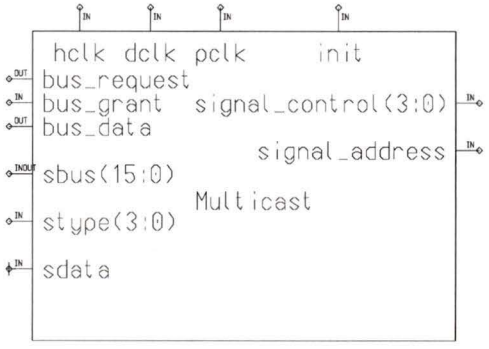


Figure 5.17 Icon of Multicast module.

5.4.1.9 Local SM

The **Local SM** module reads OAM cells at their endpoints and monitors passing non-endpoint OAM cells as indicated by the **stype** port. These cells can be used for connectivity verification and alarms surveillance. The **Local SM** module also reads headers of cells that have produced table errors, though their payloads are discarded. The processing within this module is software based, and interaction can be performed with the global **SM** as required.

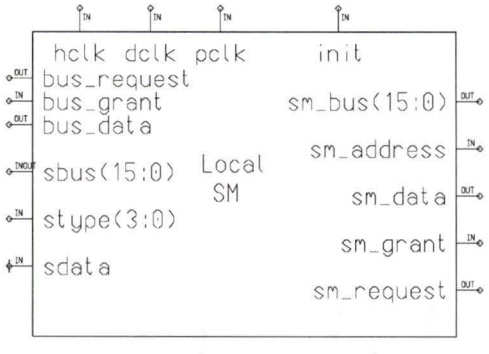


Figure 5.18 Icon of Local SM module.

5.4.2 Cell Switching Fabric

In the current implementation, the **CSF** is a TDM bus. Therefore the **SIM**s are driving one common bus within the **CSF** module which is routed to all **SOM**. The basis for arbitration within the **CSF** is the **TDM Bus Arbiter** of Figure 5.19. On a rising **pclk**, each of the **csf_request** flags are examined within the arbiter. If none has been granted, and there is sufficient time within the system clock period to transfer a cell, a request will be granted. A grant counter is used to insure that the same port does not get access to the bus twice until all other ports with cell are serviced. A grant flag is reset with the removal of the request.

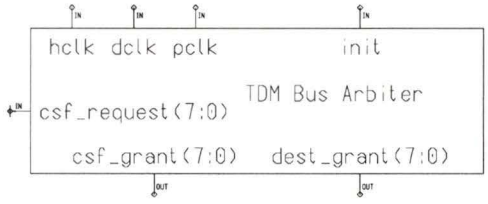


Figure 5.19 Icon of TDM Bus Arbiter module.

5.4.3 Connection Admission Control

Below the **CAC** module lies a structural architecture of two components, the **Arbiter** and the **CAC Handler**, as shown in Figure 5.20. The **Arbiter** controls access to the **cac_bus** as well as making write requests of the **SOM** for the **CAC Handler**. On each rising **pclk**, the **Arbiter** examines the **CAC Handler cell_req** flag, as well as requests from the **IM**. In the arbitration process, requests from the **CAC Handler** have priority. If such a request exists, the **Arbiter** sets the **write_req** flag, and places the desired port, provided through the **cell_address** pin, on the **address** bus. When this request is granted by the **SOM** on the **write_grant** line, the **CAC Handler** is notified, and cell transfer can begin. The **write_req** flag is reset when the **cell_req** line is lowered. If no request from the **CAC Handler** exists, the request lines of the **SIM** are polled in round robin fashion. Grants are provided by setting the grant flag and writing the address of the granted port on the address bus. This grant flag is reset with the request flag.

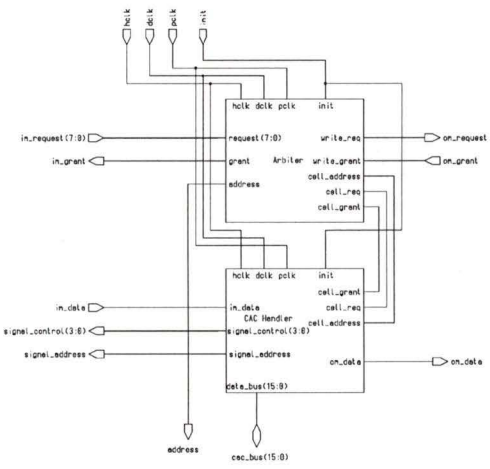


Figure 5.20 CAC block diagram.

SIM-CAC Timing

A timing diagram of the **SIM-CAC** arbitration process is shown in Figure 5.21. In this diagram, two requests are made of the **CAC**, denoted by the hexadecimal representation of the **im_cac_request** flag. The request from port 0 is processed first, followed by the request of port 1 when the first request is reset, as shown by the combination of **cac _address** and **im_cac_grant** in Figure 5.21.

The **CAC Handler** module processes the signalling cells that it receives from the **SIM** and generates signalling cells to be passed to the **SOM**. Cell reading is performed when **im_data** is set, while cell writing is performed only when the bus has been granted through **cell_grant**. Within the **CAC Handler** lies a software processor that reassembles the Q.2931 messages, processes the message and makes an admission decision, generates a new signalling message, segments the message, and forwards it to the output port. Any new additions to table within the **SIM** are made through the **signal_control** and **signal_address**. Through these pins, a new entry can be added or old entry deleted from the routing/multicast or traffic table associated with any port.

Figure 5.21 CAC arbitration timing diagram

5.4.4 System_Management

The **SM** module of Figure 5.22, like the **CAC** module, is further divided into two modules. The first is an **Arbiter** identical to that in the **CAC** module. The second is the **SM Handler** that processes OAM cells forwarded from the **SIM**, and forwards new OAM cells to the **SOM**. As the arbitration process is identical to the **CAC** arbitration of the previous section, it will not be discussed here. Only those cells that are unable to be processed in the **Local SM** of the **SIM** will be passed on for further processing in the **SM Handler**.

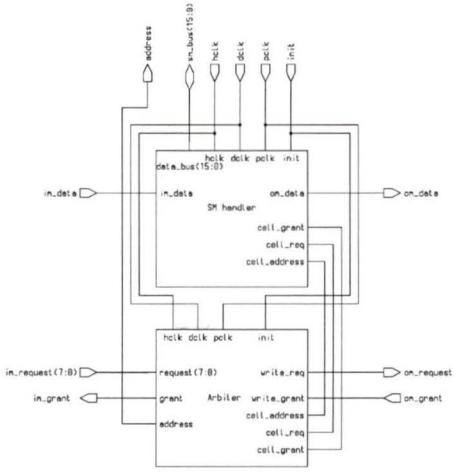


Figure 5.22 System management block diagram.

5.4.5 Output Module and Sub Output Module

As with the **Input Module**, the **Output Module** consists of a set of **SOM**, one for each port. There are no internal signals within the **Output Module**. The **SOM**, depicted in Figure 5.23, receives cells from the **CAC**, **CSF**, **SM** and the corresponding **SIM**. Here these cells are queued until a request is made, when the cell with the current highest priory is serially transmitted to the Physical layer. Structurally, each **SOM** is divided into two components as shown in Figure 5.24. In this architecture, the **Priority** module maintains the cell queues, and uses flags to indicate buffer occupancy. The **Scheduler** interacts with the **request** flag from the Physical layer, and grants write permission to the queues or flags the unassigned cell generator when a Physical layer request is received.

For signalling, OAM and ILMI cells, the **Priority** module maintains an individual dedicated queue of 8 cells. For the user cells, the **Priority** module maintains 64

shared cell buffers per port. In addition, the CLP bit is used to maintain buffer positions for high priority cells at the expense of cells tagged low priority. The shared queuing is implemented through the maintenance of address queues for each of the priority classes, and an idle address buffer indicating empty locations. The buffers themselves are shift registers that are enabled by an address in combination with a read enable, or a write enable with shift register pointer. The writing process recovers the serial cell using a ring counter in the opposite operation from the **Serpar** module of the **SIM**. The logical arrangement of the shared user buffering is shown in Figure 5.25. The basis for the shared queuing is the control module that handles enabling the read/write signals, the ring counter for writing, adding addresses to the address queues, and recirculating addresses onto the idle address buffer.

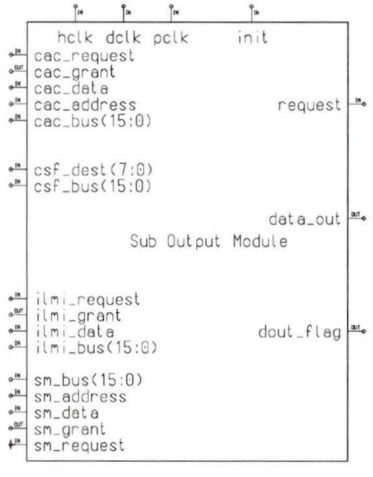


Figure 5.23 Icon of Sub Output Module.

Sub Output Module Timing

In Figure 5.26 a timing diagram for cell flow through the shared user output buffer is shown. In this diagram, two cells are written to the buffer. The first of these has a lower priority than the second. When the next request is made, the first is bypassed in favour of the second higher priority cell. In Figure 5.26, the buffering of the first cell sets the **user_flags** to 02hex, indicating cell priority of "001" (**user_flags**(1) = '1'). After the second cell is received, **user_flags** changes to 0Ahex, indicating the second cell has priority "011" (**user_flags**(3) = '1'). From the **user_grants** flags, the cell of priority "011" is given write permission first through grant 08hex, followed by the

lower priority cell through 02hex.

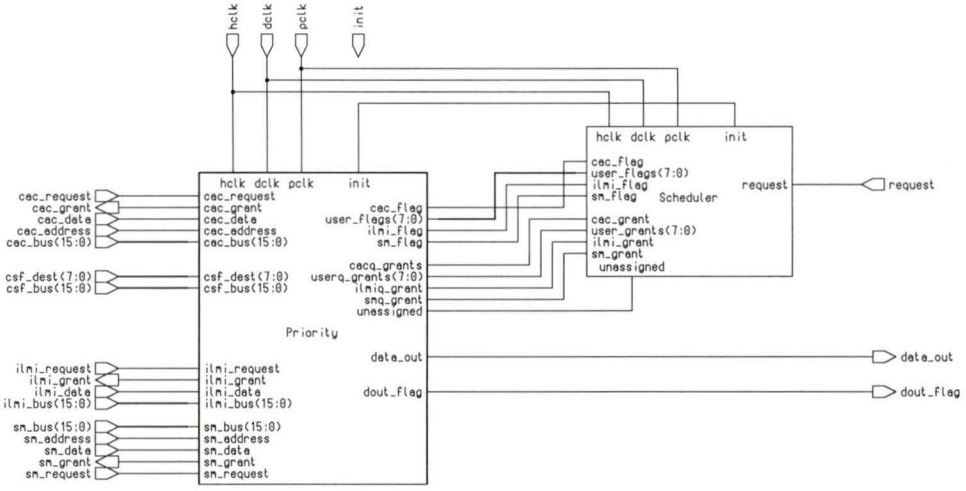


Figure 5.24 Sub Output Module block diagram.

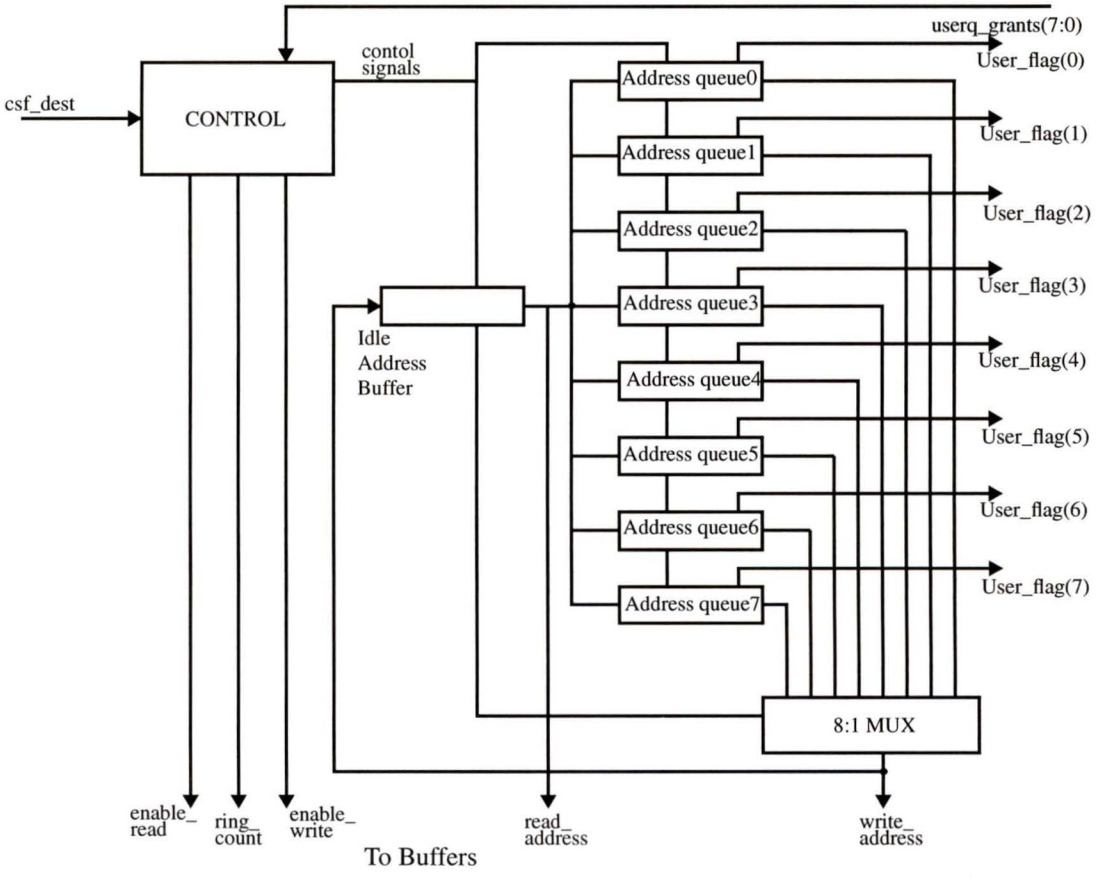


Figure 5.25 Shared user buffering block diagram.

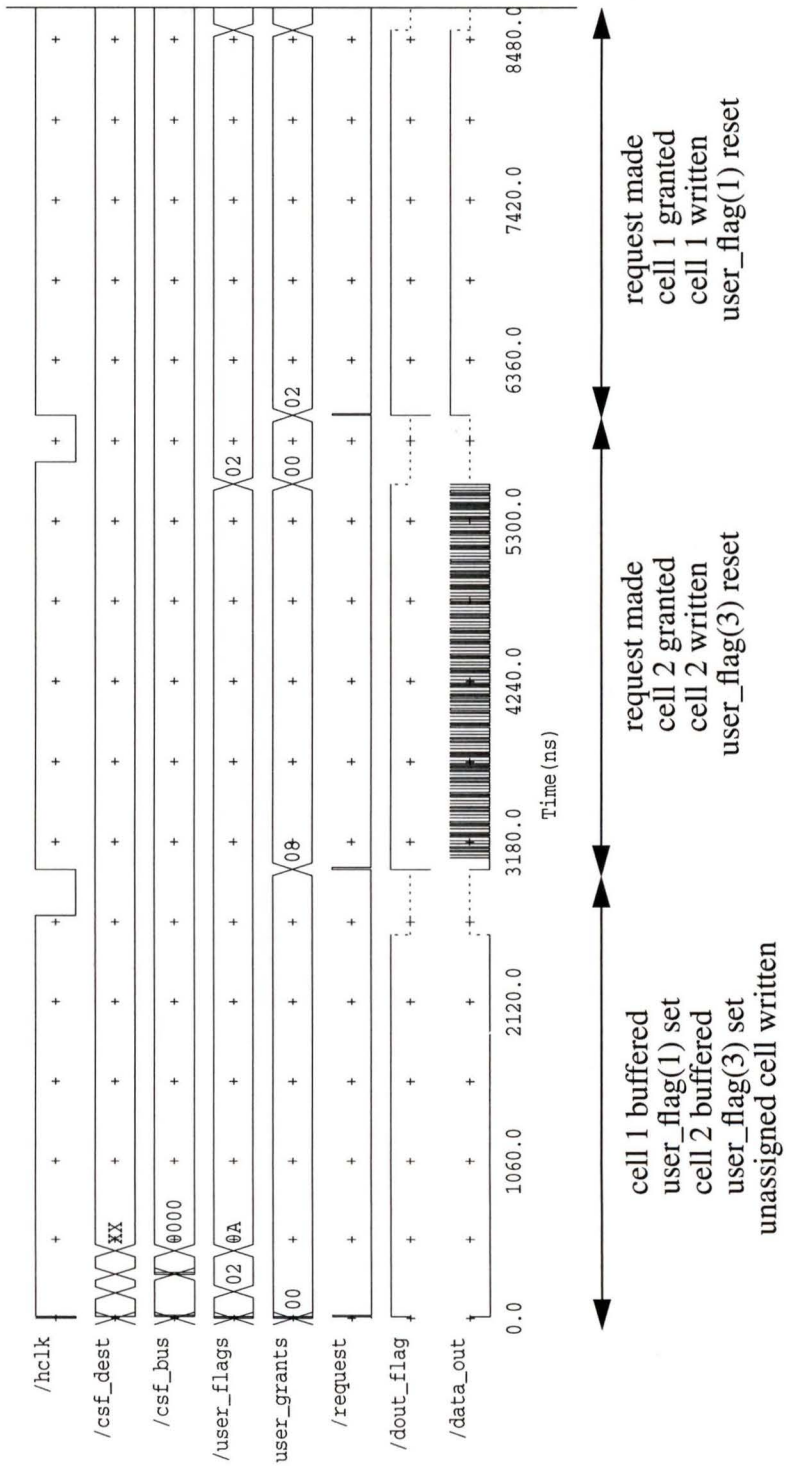


Figure 5.26 SOM timing diagram.

5.4.6 Bottom Layer Components

To provide for a structural architecture below the block diagrams of the previous sections, an RTL component library was used in addition to several purely behavioural ARCHITECTUREs. This RTL library is composed of flipflops, registers, shift registers and counters. Within the **SIM**, only the route table module is purely behavioural with its CAM composition. The other modules were broken down further into RTL components with both selected and conditional signal assignments. Within the **SOM**, only the control logic of queues was behaviourally implemented with a modular division within the priority module to implement both the different queues and the queue controllers structurally.

5.5 Simulation

Further details on simulation methods can be found in Appendix B.

This design was simulated to verify its logic. This primarily included the flow of cells across the switch and the necessary resources in terms of flags and buses to accomplish the requirements of an ATM layer switch. For simulation purposes, *delta delays* were used for signal propagation.

The first stage of simulation verified the logic at a strictly *behavioural* level utilizing PROCESSes and *behavioural* language constructs. As a second stage, *structural* models were used instead of the final stage of RTL descriptions for base components. It was found that simulation at this level was tedious, as every *structural* layer increased the simulation time dramatically, and that memory requirements were beyond the resources being used. For this reason, the final results use RTL descriptions.

The full switch was simulated for multiple cells being received over several **hclk** periods to verify the performance of the **SIM**, **CSF** and **SOM**. Additionally the **SIM** and **SOM** were simulated individually to examine the handling of different cell types, and the priority queuing discipline. Shared queuing was simulated in depth to examine the recirculation of idle addresses and the address queues for individual traffic classes. The bus arbitration schemes for the **CAC** and **SM** modules was also verified.

5.6 Delay Analysis

As *delta delays* were used to perform simulation, the logic of the switch with actual rise, fall and delay times was not simulated, which is the next step in the design process. How-

ever, by examining the necessary gate delays to enable registers and the read time for fast CAMs, an idea of the necessary device speed can be gained.

For the cell sort module of the **SIM**, the cell type decision must be made before **hclk** falls. The major constraint at this point lies in the table lookup. However, an examination of a CAM intended for table lookup applications revealed a 64-bit table capable of access times of 100ns [28], which is within the time constraints including header transfer time and other combinational logic within the cell sort module if the SONET 155.52 Mb/s rate is assumed. The other constraint lies in enabling the serial to parallel shift registers of the **Serpar** module, which requires 2 AND/OR gate delays plus 1 flipflop delay. In the **SOM**, the delay constraints also lie in the enabling of the shift register buffers, which have gate delays equal to **Serpar** module. By examining gate delays for high-speed CMOS technology, it is estimated that current device delays are within the limits of this design.

5.7 Summary

In this chapter, a more detailed description of the ATM layer switch outlined in Chapter 4 has been given. This switch provides a hardware framework for a full implementation that includes software-based modules for signalling and management. The switch was built using a hierarchical VHDL design that divided the functions into small components composed of RTL base components and small behavioural architectures. The design was simulated to verify the logic and the cell flow across the switch.

Chapter 6

Conclusions and Future Work

6.1 Thesis Conclusions and Contributions

An ATM switch is composed of more than just a switching fabric. A full implementation supports modules for signalling and management as well as providing for traffic contract enforcement and other ATM layer functions.

The requirements of an ATM switch have been examined to develop a model for a full implementation of an ATM layer switch. This model incorporates the required functions of the switch at the UNI, as well as other functions that enhance switch performance.

Using high-level VHDL modelling and top-down structured VLSI design methodology, an ATM switch has been implemented that performs many of the functions of the ATM layer of the Protocol Model while providing a hardware framework capable of incorporating software-based modules for signalling, management and traffic control. This implementation is highly modular.

The functions of the ATM layer switch have been decomposed into small, manageable components to provide for a simplified design process, and easy design changes. The main modules of the design have been designed to be as self-contained as possible with well-defined interfaces to make them transportable between designs.

This implementation is a hybrid input-output buffered TDM bus switch. The implementation supports multicasting, ILMI, signalling and management cells in addition to user flows. Eight classes of user cells are supported in addition to the priority indicated by the CLP-bit, and shared buffering is used on an output port basis to minimize the cell loss for user cells. A priority scheduling algorithm is used at the output port to insure the

QoS of the connections, and this scheduler is left independent of cell buffering to allow for the implementation of different algorithms. This implementation provides the means to monitor OAM cell flow and provision switched connections in addition to permanent connections. The results of UPC traffic enforcement can also be applied in terms of cell tagging and cell discard.

This switch was simulated to verify its logic and resource allocation schemes. The switch performance has been verified for multiple cell arrivals and types using cell scheduling over several system clock periods.

6.2 Suggested Future Work

The next step in the design process requires the implementation of a low-level technology-based switch to study the gate delays and the effect of driving capacitances on the switch performance. The majority of the components, as they are composed of a set of RTL base-components, can be modified by altering the structural components used in the lowest layer of the hierarchical design.

Additionally, the software-based modules must be programmed for incorporation within the switch architecture. These modules include Connection Admission Control, System Management and UPC enforcement. The implementation of these modules involves further development of a model for ATM traffic to make connection admission decisions, and an implementation of the Generic Cell Rate Algorithm as defined in standard documents.

Also, as the next generation of ATM Forum standards are soon to be released, these new standards must be examined so new functions can be implemented as needed.

Bibliography

[1] M. de Prycker, *Asynchronous Transfer Mode*, Second Edition, Ellis Horwood Limited, UK, 1993.

[2] D. McDysan and D. Spohn, *ATM: Theory and Application*, McGraw-Hill, Inc., USA, 1995.

[3] H. Saito, *Teletraffic Technologies in ATM*, Artech House, USA, 1994.

[4] A. Acampora, *An Introduction to Broadband Networks*, Plenum Press, New York, 1994.

[5] M. Montpetit and M. Cote, "LAN interconnection by onboard switching satellites", *Proceedings of the Canadian Conference of Electrical and Computer Engineering*, Montreal, Que., pp. 145-147, 1995.

[6] J. Terry, "Alternative technologies and delivery systems for broadband ISDN system access", *IEEE Communications Magazine*, vol. 30, pp. 58-64, August 1992.

[7] R. Gejji, "Mobile multimedia scenario using ATM and microcellular technologies", *IEEE Trans. Vehic. Tech.*, vol. 43, pp. 699-703, August 1994.

[8] *ATM User-Network Specification*, Version 3.1, The ATM Forum, 1994.

[9] Y. Yeh, M. Hluchyj, and A. Acampora, "The knockout switch: A simple, modular architecture for high-performance packet switching", *IEEE J. Select. Areas Commun.*, vol. SAC-5, pp. 1274-1282, October 1987.

[10] P. Newman, "ATM technology for corporate networks", *IEEE Communications Magazine*, vol. 30, pp. 90-101, April, 1992.

[11] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division packet switch", *IEEE Trans. Comm.*, vol. COM-35, pp. 1347-1356, December 1987.

[12] M. Hluchyj, and M. Karol, "Queueing in high-performance packet switching", *IEEE J. Select. Areas Commun.*, vol. 6, pp. 1587-1597, December 1988.

[13] J. Garcia-Haro and A. Jajszcyk, "ATM Shared-memory switching architectures", *IEEE Network*, vol. 8, pp. 27-40, July/August 1994.

[14] E. Zegura, "Architectures for ATM switching systems", *IEEE Communications Magazine*, vol. 31, pp. 28-37, February 1993.

[15] C. Fayet, A. Jacques, and G. Pujolle, "High speed switching for ATM: the BSS", *Computer Networks and ISDN Systems*, vol. 26, pp. 1225-1233, May 1994.

[16] W. Denzel, A. Engbersen and I. Iliadis, "A flexible shared-buffer switch for ATM at Gb/s rates", *Computer Networks and ISDN Systems*, vol. 27, pp. 611-624, January 1995.

[17] H. Ahmadi and W. Denzel, "A survey of modern high-performance switching techniques", *IEEE J. Select. Areas Commun.*, vol. 7, pp. 1091-1103, September 1989.

[18] H. Kim, "Design and performance of Multinet switch: a multistage ATM switch architecture with partially shared buffers", *IEEE/ACM Trans. Networking*, vol. 2., pp. 571-580, December 1994.

[19] R. Bianchini and H. Kim, "The Tera project: a hybrid queuing ATM switch architecture for LAN", *IEEE J. Select. Areas Commun.*, vol. 13, pp. 673-685, May 1995.

[20] T. Chen and S. Liu, "Management and control functions in ATM switching systems", *IEEE Network*, vol. 8, pp. 27-39, July/August 1994.

[21] R. Black, I. Leslie, D. McAuley, "Experiences of building an ATM switch for the local area", *Proceedings of SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, London, England, pp. 158-167, 1994.

[22] J. Cox Jr., M. Gaddis, and J. Turner, "Project Zeus", *IEEE Network*, vol.7, pp. 20-30, March 1993.

[23] M. Collivignarelli, A. Daniele, P. De Nicola, L. Licciardi, M. Torolla, and A. Zappalorto, "A complete set of VLSI circuits for ATM switching", *1994 IEEE Globecom Conference Record*, San Franciso, Cal., pp. 134-138, 1994.

[24] C. Mead and L. Conway, *Introduction to VLSI System Design*, Addison-Wesley Publishing Company, USA, 1980.

[25] N. Weste and K. Eshragian, *Principles of CMOS VLSI Design: A Systems Perspective*, Addison-Wesley Publishing Company, Don Mills, 1985.

[26] D. Perry, *VHDL*, Second Edition, McGraw-Hill, Inc., USA, 1994.

[27] B. Prince, *Semiconductor Memories*, Second Edition, John Wiley & Sons Ltd., England, 1991.

[28] A. McAuley and C. Cotton, "A self-testing reconfigurable CAM", *IEEE J. Solid-State Circuits*, vol. 26, pp. 257-261, March 1991.

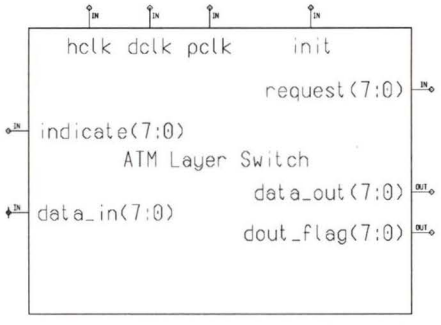
[29] J. Gilderson, F. El-Guibaly, and V. K. Bhargava, "*VHDL ATM Switch Design*", Technical Report VMC/2, Dept. of Electrical and Computer Engineering, University of Victoria, 1995.

Appendix A

VHDL Switch Schematics

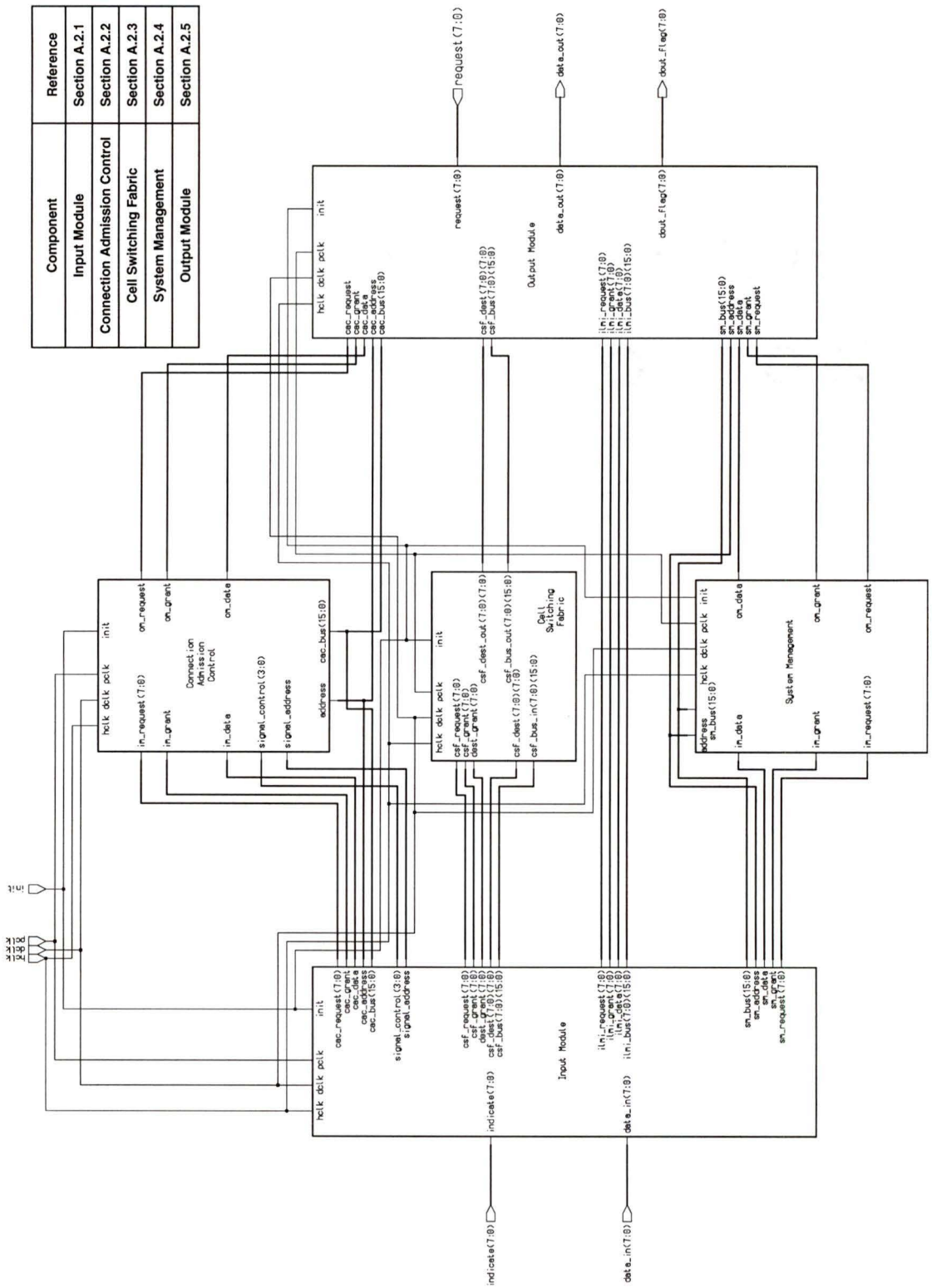
In Chapter 5, an implementation of an ATM switch in VHDL was described. This description provided functional details of the key components that compose this implementation. In this Appendix, schematics for an 8-port implementation are provided, as are port descriptions for each component.

A.1 Top Level : ATM Layer Switch



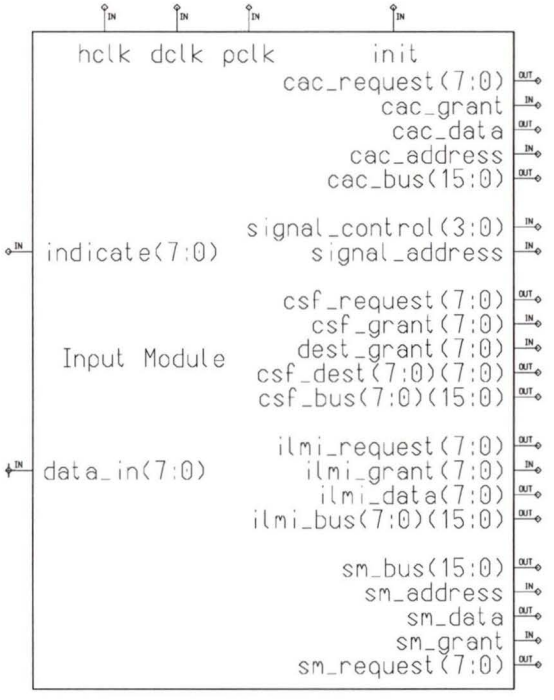
PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	intialization signal
indicate(7:0)	IN	set of flags indicating cell arrival
data_in(7:0)	IN	set of serial data lines for cell reading
request(7:0)	IN	set of cell request flags
data_out(7:0)	OUT	set of serial data lines for cell writing
dout_flag(7:0)	OUT	set of flags indicating cell departure

A.1.1 ATM Layer Switch Structural Architecture



A.2 Top Layer Structural Component

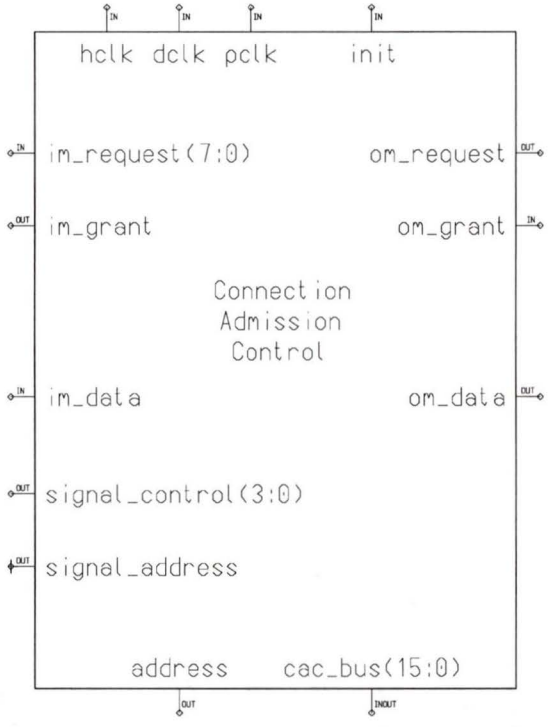
A.2.1 Input Module



PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
indicate(7:0)	IN	set of flags indicating cell arrival
data_in(7:0)	IN	set of serial data lines for cell reading
cac_request(7:0)	OUT	set of request flags for CAC bus
cac_grant	IN	grant flag for CAC bus
cac_data	OUT	flag indicating data on CAC bus from IM
cac_address	IN	address of port granted CAC bus
cac_bus(15:0)	OUT	data bus for transfer of signalling cells
signal_control(3:0)	IN	control/data lines for manipulation of tables
signal_address	IN	port address of table to be manipulated
csf_request(7:0)	OUT	set of flags requesting access to CSF
csf_grant(7:0)	IN	set of grant flags for CSF data transfer access
dest_grant(7:0)	IN	set of grant flags for CSF destination indication
csf_dest(7:0)(7:0)	OUT	set of destination indications
csf_bus(7:0)(15:0)	OUT	set of cell data buses
ilmi_request(7:0)	OUT	set of request flags to transfer ILMI cells

PIN	DIRECTION	COMMENTS
ilmi_grant(7:0)	IN	set of grant flags for ILMI cell transfer
ilmi_data(7:0)	OUT	set of flags indicating data on ILMI data bus
ilmi_bus((7:0)(15:0)	OUT	set of data buses for ILMI cells
sm_bus(15:0)	OUT	data bus for transfer of management cells
sm_address	IN	address of port granted SM bus
sm_data	OUT	flag indicating data on SM bus from IM
sm_grant	IN	grant flag for SM bus
sm_request(7:0)	OUT	set of request flags for SM bus

A.2.2 Connection Admission Control

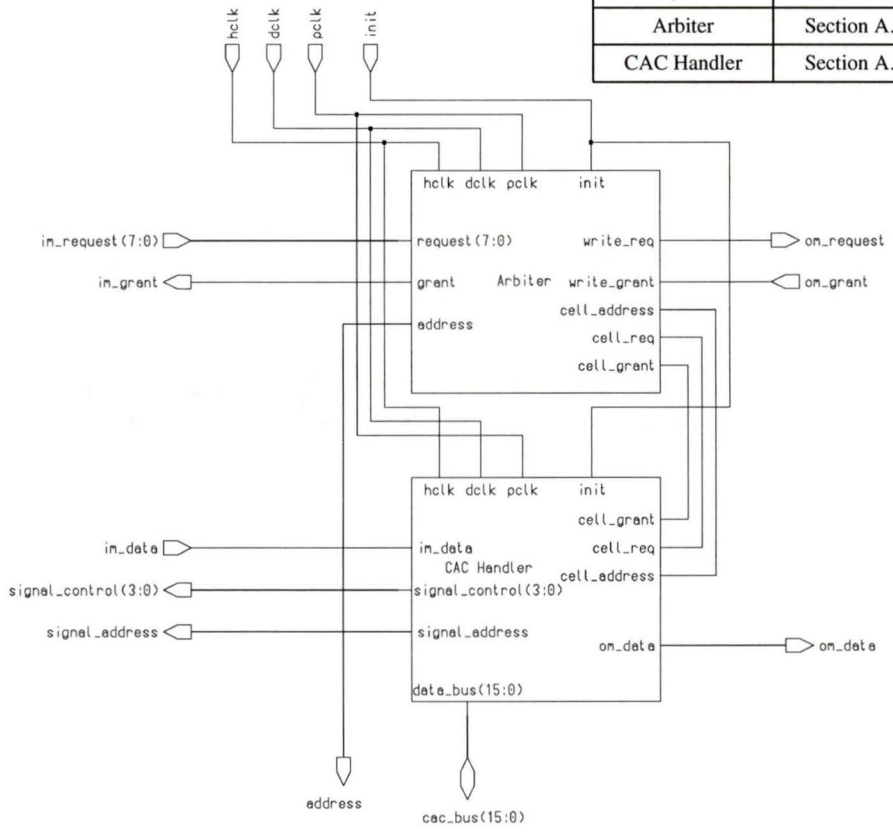


PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
im_request(7:0)	IN	set of request signals from IM for CAC bus
im_grant	OUT	grant flag to IM for CAC bus
im_data	IN	flag indicating data on CAC bus from IM
signal_control(3:0)	OUT	control/data lines for SIM table manipulation
signal_address	OUT	port address of table to be manipulated
om_request	OUT	request flag for cell transfer to SOM
om_grant	IN	grant flag for cell transfer to SOM

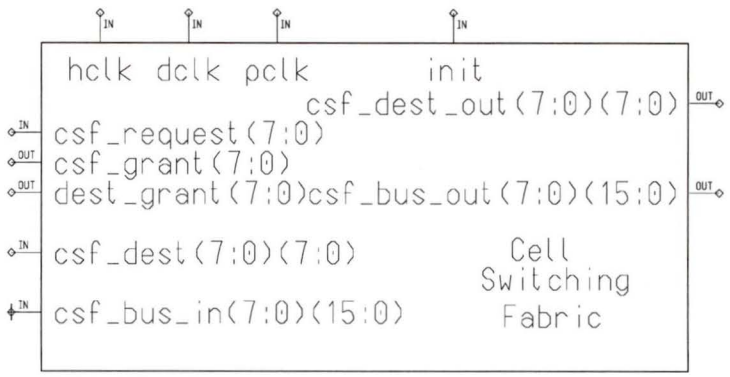
PIN	DIRECTION	COMMENTS
om_data	OUT	flag indicating data on CAC bus for SOM
address	OUT	address of port: SIM or SOM
cac_bus	INOUT	data bus for signalling cells

A.2.2.1 Connection Admission Control Structural Architecture

Component	Reference
Arbiter	Section A.4.2
CAC Handler	Section A.4.1

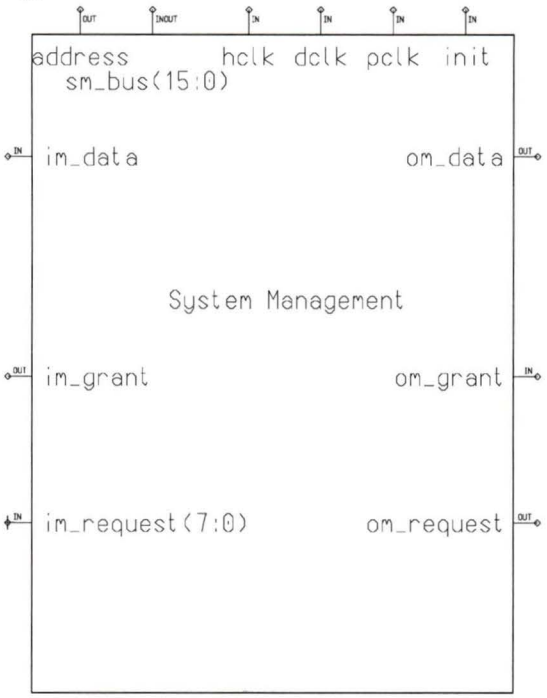


A.2.3 Cell Switching Fabric



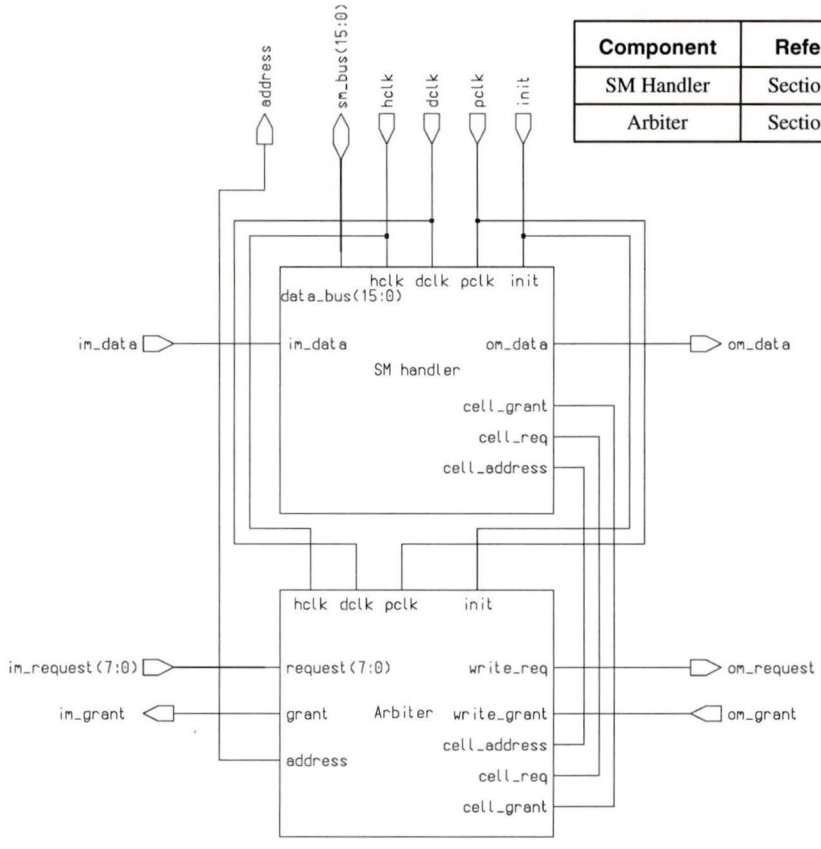
PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signals
csf_request(7:0)	IN	set of request flags from IM for CSF access
csf_grant(7:0)	OUT	set of grant lines to IM for cell transfer
dest_grant(7:0)	OUT	set of grant lines to IM for destination indication
csf_dest(7:0)(7:0)	IN	set of destination indications
csf_bus_in(7:0)(15:0)	IN	set of cell data buses
csf_dest_out(7:0)(7:0)	OUT	set of destination indications
csf_bus_out(7:0)(15:0)	OUT	set of cell data buses

A.2.4 System Management



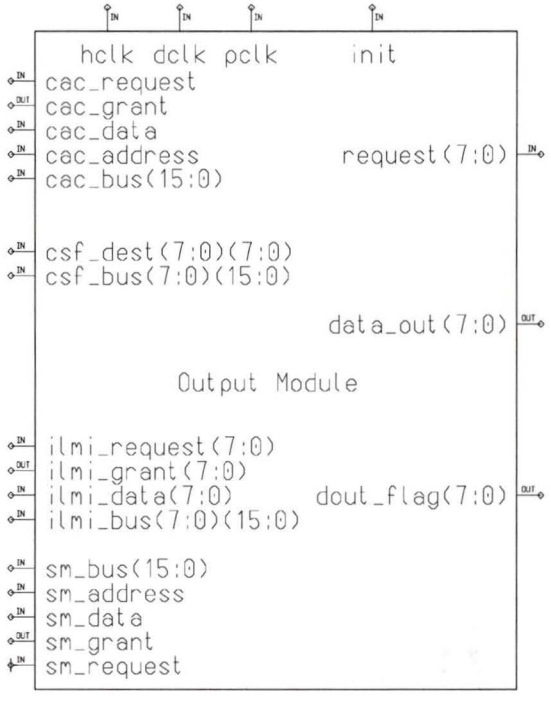
PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signals
address	OUT	address of port: SIM or SOM
sm_bus(15:0)	INOUT	data bus for management cells
im_data	IN	flag indicating data on SM bus from IM
im_grant	OUT	grant flag to IM for SM bus
im_request(7:0)	IN	request flags from IM for SM bus
om_data	OUT	flag indicating data on SM bus from SM for OM
om_grant	IN	request flag for cell transfer to SOM
om_request	OUT	grant flag for cell transfer to SOM

A.2.4.1 System Management Structural Architecture



Component	Reference
SM Handler	Section A.6.1
Arbiter	Section A.4.2

A.2.5 Output Module

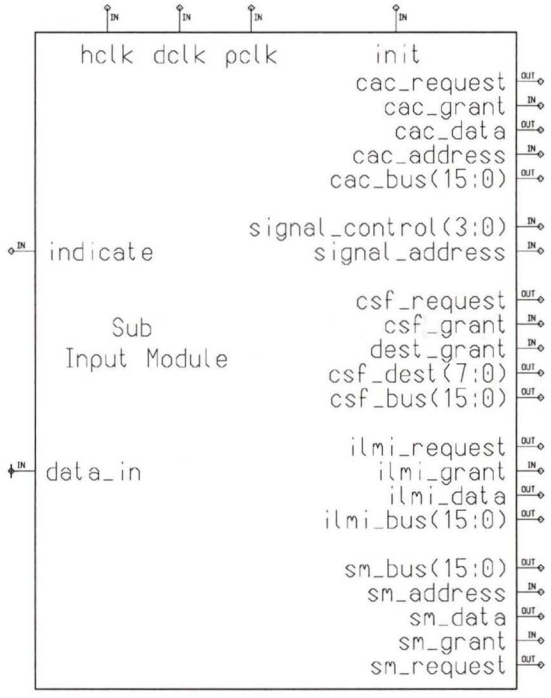


PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
cac_request	IN	request flag from CAC for cell transfer
cac_grant	OUT	grant flag to CAC for cell transfer
cac_data	IN	flag indicating data on CAC bus for OM
cac_address	IN	port address for data on CAC bus
cac_bus(15:0)	IN	data bus for signalling cells
csf_dest(7:0)(7:0)	IN	set of destination indications
csf_bus(7:0)(15:0)	IN	set of cell data buses
ilmi_request(7:0)	IN	set of request flags to transfer ILMI cells
ilmi_grant(7:0)	OUT	set of grant flags for ILMI cell transfer
ilmi_data(7:0)	IN	set of flags indicating data on ILMI bus
ilmi_bus(7:0)(15:0)	IN	set of data buses for ILMI cells
sm_bus(15:0)	IN	data bus for management cells
sm_address	IN	port address for data on SM bus
sm_data	IN	flag indicating data on SM bus for OM
sm_grant	OUT	grant flag to SM for cell transfer
sm_request	IN	request flag from SM for cell transfer
request(7:0)	IN	set of cell request flags

PIN	DIRECTION	COMMENTS
data_out(7:0)	OUT	set of serial data lines for cell writing
dout_flag(7:0)	OUT	set of flags indicating cell departure

A.3 Input Module Components

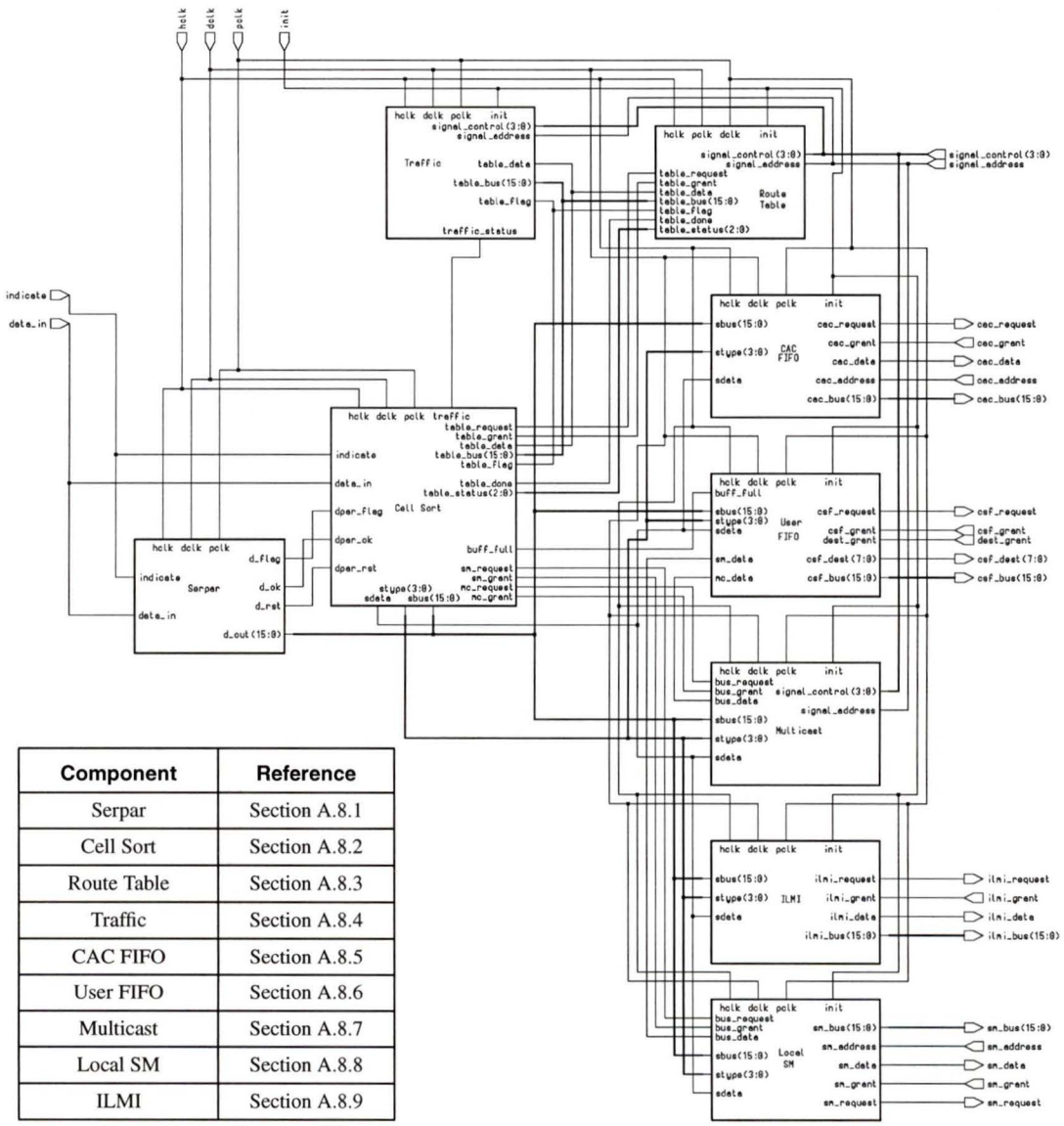
A.3.1 Sub Input Module



PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
indicate	IN	flag indicating cell arrival
data_in	IN	serial data line for cell reading
cac_request	OUT	request flag for CAC bus
cac_grant	IN	grant_flag for CAC bus
cac_data	OUT	flag indicating data on CAC bus from SIM
cac_address	IN	address of port granted CAC bus
cac_bus(15:0)	OUT	data bus for signalling cells
signal_control(3:0)	IN	control/data lines for table manipulation
signal_address	IN	port address of table to be manipulated
csf_request	OUT	flag requesting access to CSF

PIN	DIRECTION	COMMENTS
csf_grant	IN	grant flag for CSF access
dest_grant	IN	grant flag for destination indication
csf_dest(7:0)	OUT	destination indication
csf_bus(15:0)	OUT	cell data bus
ilmi_request	OUT	request flag to transfer ILMI cell
ilmi_grant	IN	grant flag for ILMI cell transfer
ilmi_data	OUT	flag indicating data on ILMI data bus
ilmi_bus(15:0)	OUT	ILMI data bus
sm_bus(15:0)	OUT	data bus for management cells
sm_address	IN	address of port granted SM bus
sm_data	OUT	flag indicating data on SM bus from SIM
sm_grant	IN	grant flag for SM bus
sm_request	OUT	request flag for SM bus

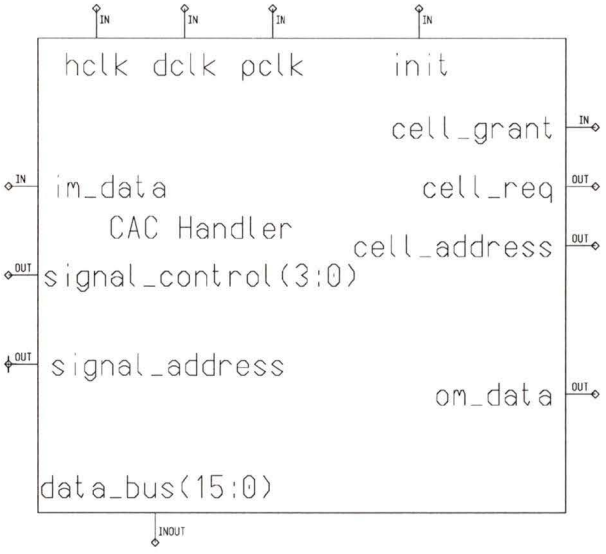
A.3.1.1 Sub Input Module Structural Architecture



Component	Reference
Serpar	Section A.8.1
Cell Sort	Section A.8.2
Route Table	Section A.8.3
Traffic	Section A.8.4
CAC FIFO	Section A.8.5
User FIFO	Section A.8.6
Multicast	Section A.8.7
Local SM	Section A.8.8
ILMI	Section A.8.9

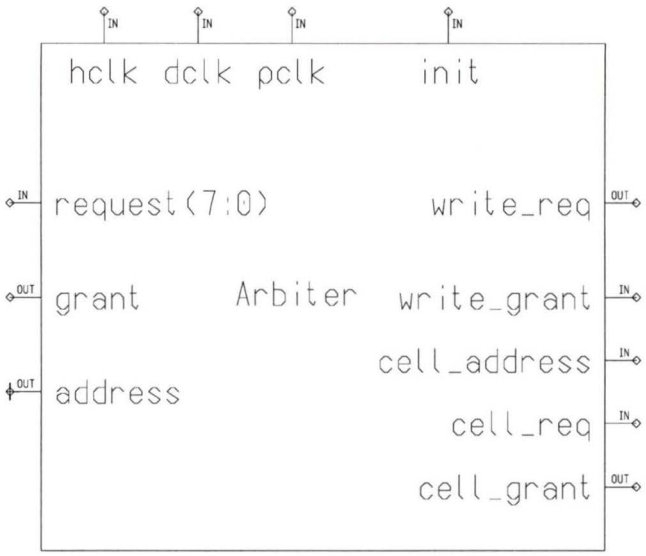
A.4 Connection Admission Control Components

A.4.1 CAC Handler



PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
im_data	IN	flag indicating data on CAC bus from IM
signal_control(3:0)	OUT	control/data lines for SIM table manipulation
signal_address	OUT	port address of table to be manipulated
cell_grant	IN	grant flag from arbiter for CAC bus
cell_req	OUT	request flag to arbiter for CAC bus
cell_address	OUT	port address to be written to
om_data	OUT	flag indicating data on CAC bus for OM
data_bus(15:0)	INOUT	data bus for signalling cells

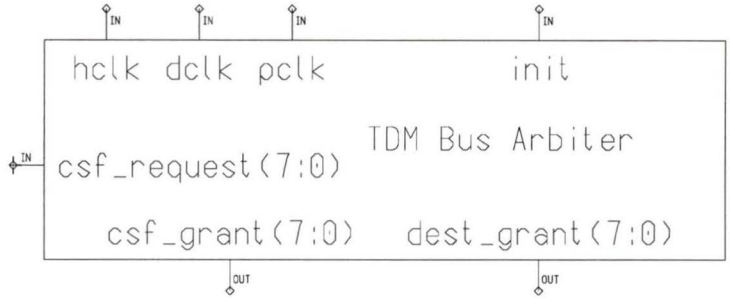
A.4.2 Arbiter



PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
request(7:0)	IN	set of request flags for bus access
grant	OUT	grant flag for bus access
address	OUT	port address granted bus access
write_req	OUT	request flag for writing
write_grant	IN	grant flag for writing
cell_address	IN	port address of handler request
cell_req	IN	handler request for writing
cell_grant	OUT	grant flag for handler request

A.5 Cell Switching Fabric Components

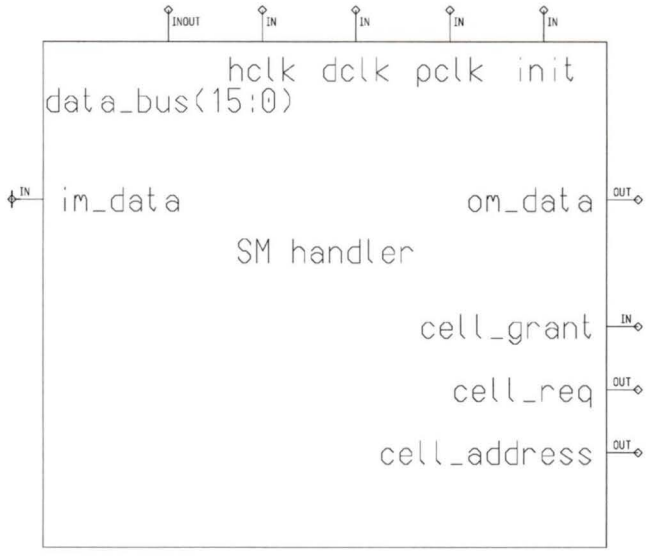
A.5.1 TDM Bus Arbiter



PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
csf_request(7:0)	IN	set of request flags for CSF access
csf_grant(7:0)	OUT	set of grant flags for CSF access
dest_grant(7:0)	OUT	set of grant flags for destination indications

A.6 System Management Components

A.6.1 SM Handler



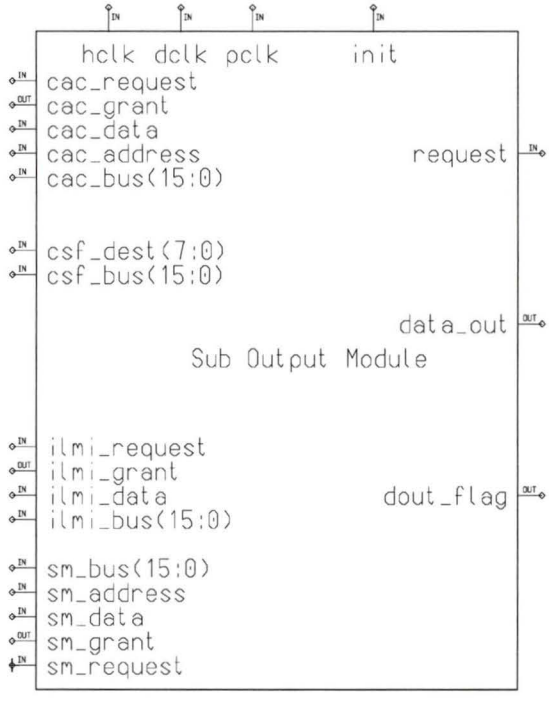
PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
data_bus(15:0)	INOUT	data bus for management cells
im_data	IN	flag indicating data on SM bus from IM
om_data	OUT	flag indicating data on SM bus for OM
cell_grant	IN	grant flag from arbiter for SM bus
cell_req	OUT	request flag to arbiter for SM bus
cell_address	OUT	port address to be written to

A.6.2 Arbiter

See Section A.4.2.

A.7 Output Module Components

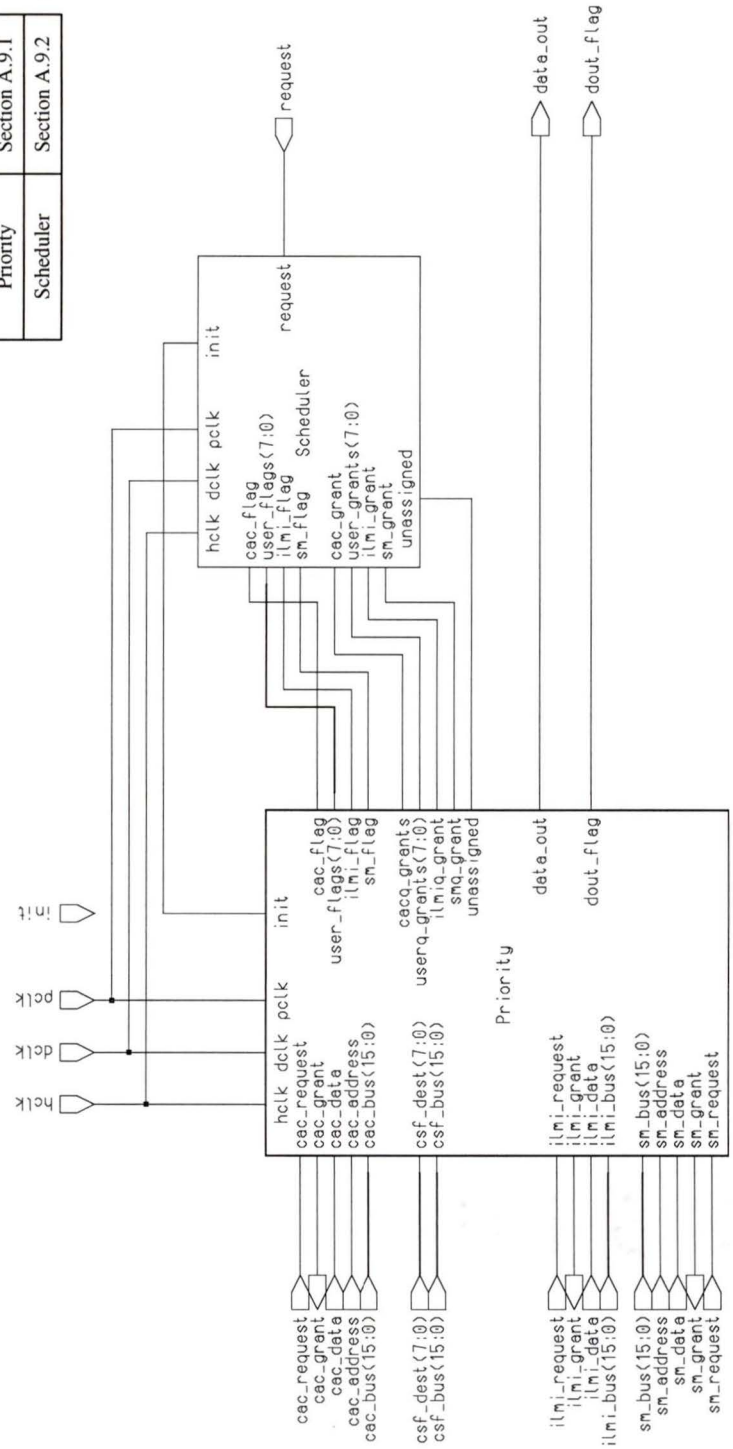
A.7.1 Sub Output Module



PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
cac_request	IN	request flag from CAC for cell transfer
cac_grant	OUT	grant flag to CAC for cell transfer
cac_data	IN	flag indicating data on CAC bus from OM
cac_address	IN	port address for data on CAC bus
cac_bus(15:0)	IN	data bus for signalling cells
csf_dest(7:0)	IN	destination indication
csf_bus(15:0)	IN	cell data bus
ilmi_request	IN	request flag to transfer ILMI cell
ilmi_grant	OUT	grant flag for ILMI cell transfer
ilmi_data	IN	flag indicating data on ILMI bus
ilmi_bus(15:0)	IN	data bus for ILMI cells
sm_bus(15:0)	IN	data bus for management cells
sm_address	IN	port address for data on SM bus
sm_data	IN	flag indicating data on SM bus for OM
sm_grant	OUT	grant flag to SM for cell transfer
sm_request	IN	request flag from SM for cell transfer
request	IN	cell request flag
data_out	OUT	serial data line for cell writing
dout_flag	OUT	flag indicating cell departure

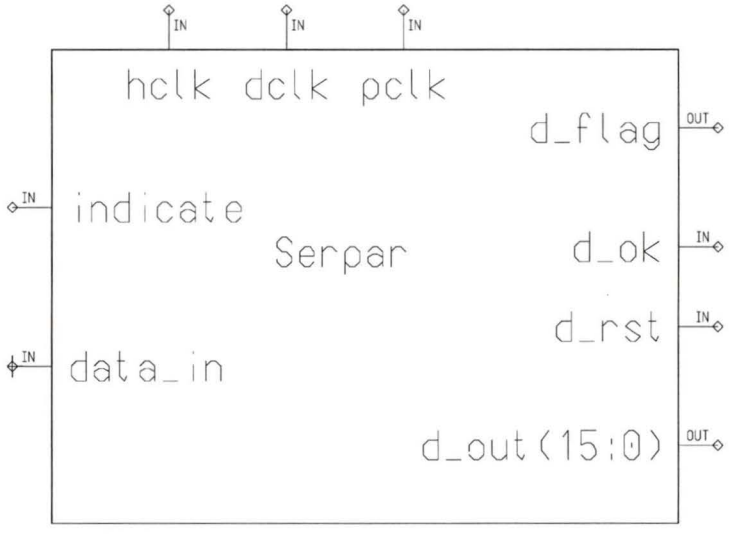
A.7.1.1 Sub Output Module Structural Architecture

Component	Reference
Priority	Section A.9.1
Scheduler	Section A.9.2



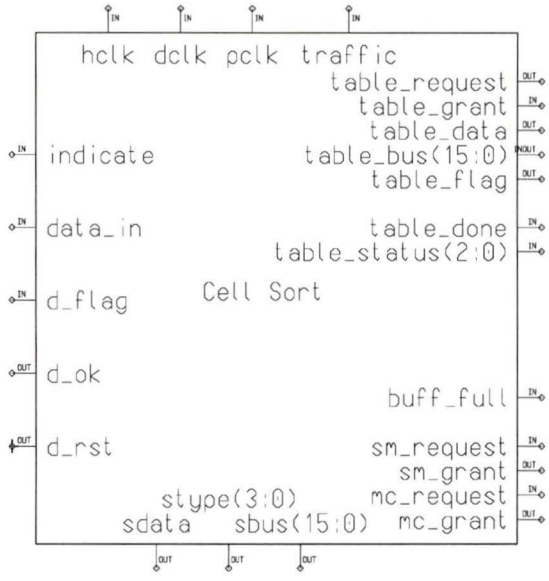
A.8 Sub Input Module Components

A.8.1 Serpar



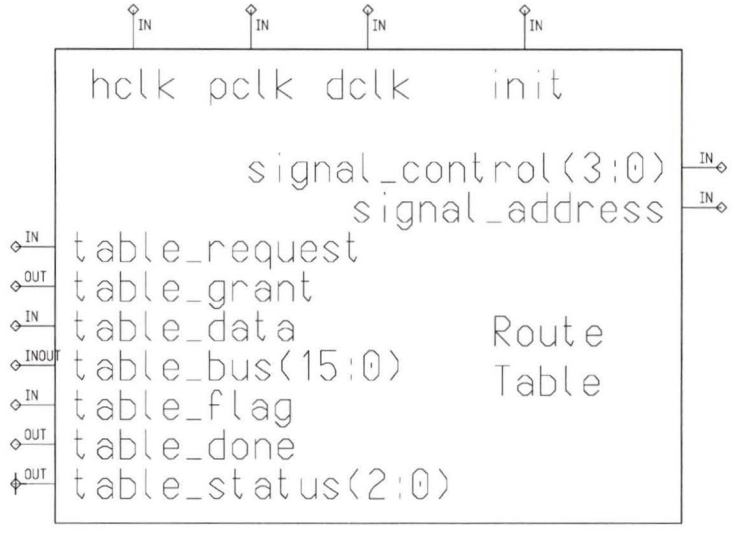
PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
indicate	IN	flag indicating cell arrival
data_in	IN	serial data line for cell reading
d_flag	OUT	flag indicating cell presence
d_ok	IN	flag granting bus access
d_rst	IN	flag indicating cell discard
d_out(15:0)	OUT	data bus for cell transfer

A.8.2 Cell Sort



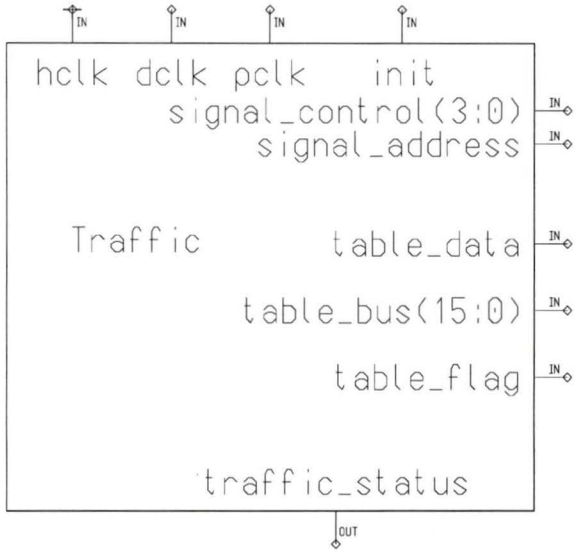
PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
traffic	IN	results of traffic contract enforcement
indicate	IN	flag indicating cell arrival
data_in	IN	serial data line for cell transfer
d_flag	IN	flag indicating cell presence in serpar
d_ok	OUT	flag granting access to serpar
d_rst	OUT	flag resetting serpar
table_request	OUT	request flag for header transfer to routing table
table_grant	IN	grant flag for header transfer to routing tabl
table_data	OUT	flag indicating header on table_bus
table_bus(15:0)	INOUT	data bus for header transfer
table_flag	OUT	flag indicating cell type
table_done	IN	flag indicating table search done
table_status(2:0)	IN	flag indicating table search results
buff_full	IN	flag inidcating status of user FIFO
sm_request	IN	request flag from l Local SM
sm_grant	OUT	grant flag from Local SM
mc_request	IN	request flag from multicast
mc_grant	OUT	grant flag to multicast
sdata	OUT	flag indicating data on sbus
stype(3:0)	OUT	flag indicating cell type on sbus
sbus(15:0)	OUT	data bus for cell transfer to SIM outputs

A.8.3 Route Table



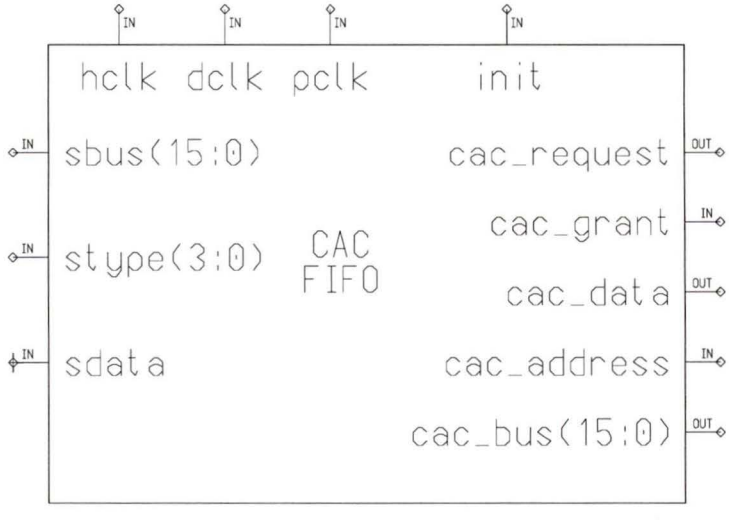
PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
table_request	IN	request flag for header transfer from cell sort
table_grant	OUT	grant flag for header transfer from cell sort
table_data	IN	flag indicating data on table_bus
table_bus(15:0)	INOUT	data bus for header transfer
table_flag	IN	flag indicating cell type
table_done	OUT	flag indicating table search done
table_status(2:0)	OUT	flag indicating table search results
signal_control(3:0)	IN	control/data lines for table manipulation
signal_address	IN	port address of table to be manipulated

A.8.4 Traffic



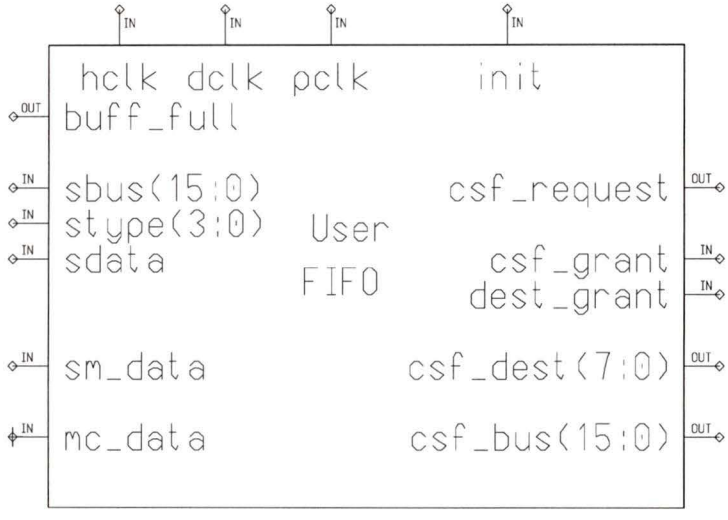
PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
signal_control(3:0)	IN	control/data lines for table manipulation
signal_address	IN	port address of table to be manipulated
table_data	IN	flag indicating header on table_bus
table_bus(15:0)	IN	data bus for header transfer
table_flag	IN	flag indicating cell type
traffic_status	OUT	results of traffic contract enforcement

A.8.5 CAC FIFO



PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
sbus(15:0)	IN	data bus for cell transfer to SIM outputs
stype(3:0)	IN	flag indicating cell type on sbus
sdata	IN	flag indicating data on sbus
cac_request	OUT	request flag for CAC bus
cac_grant	IN	grant flag for CAC bus
cac_data	OUT	flag indicating data on CAC bus
cac_address	OUT	address of port granted CAC bus
cac_bus(15:0)	OUT	data bus for transfer of signalling cells

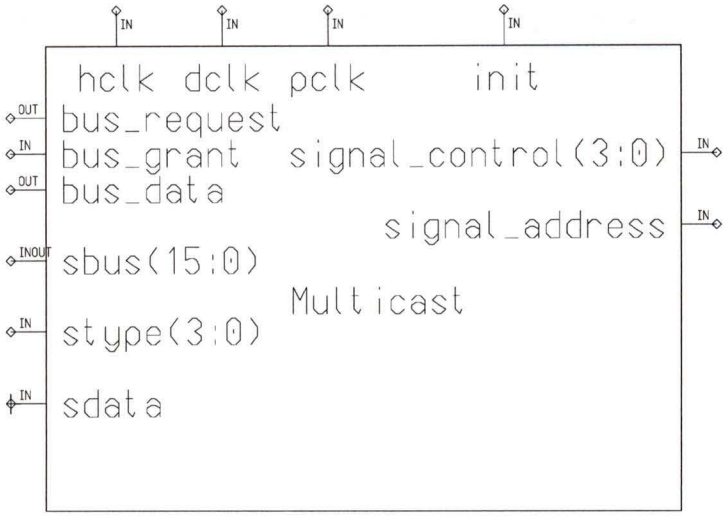
A.8.6 User FIFO



PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
buff_full	OUT	flag indicating User FIFO full
sbus(15:0)	IN	data bus for cell transfer to SIM outputs
stype(3:0)	IN	flag indicating cell type on sbus
sdata	IN	flag indicating data on sbus
sm_data	IN	flag indicating data from Local SM on sbus
mc_data	IN	flag indicating data from Local Multicast on sbus
csf_request	OUT	request flag for CSF access
csf_grant	IN	grant flag for CSF access
dest_grant	IN	grant flag for destination indication
csf_dest(7:0)	OUT	destination indication

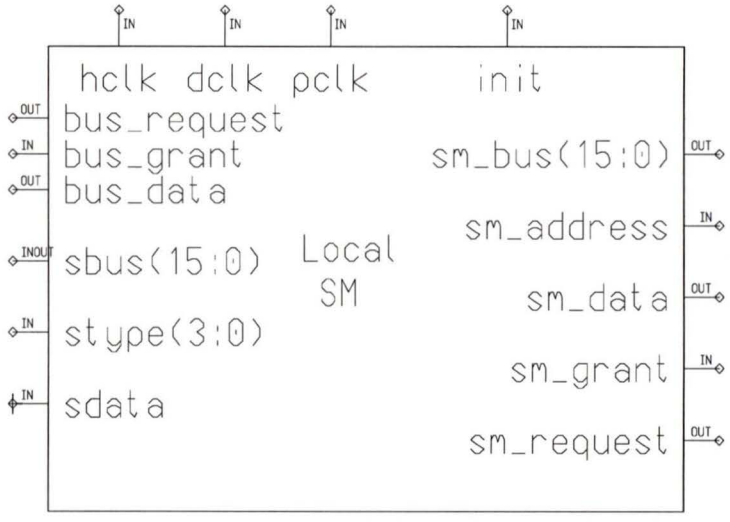
PIN	DIRECTION	COMMENTS
csf_bus(15:0)	OUT	data bus for cell transfer

A.8.7 Multicast



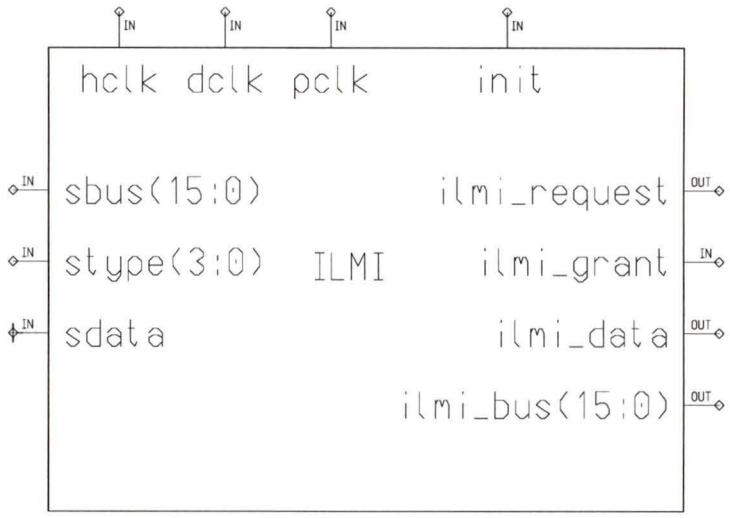
PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
bus_request	OUT	requestf flag for access to sbus
bus_grant	IN	grant flag for access to sbus
bus_data	OUT	flag indicating data on sbus
sbus(15:0)	INOUT	bus for cell transfer
stype(3:0)	IN	flag indicating type of incoming cell
sdata	IN	flag indicating incoming cell on sbus
signal_control(3:0)	IN	control/data lines for table manipulation
signal_address	IN	prot address of table to be manipulated

A.8.8 Local SM



PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
bus_request	OUT	request flag for bus access
bus_grant	IN	grant flag for bus access
bus_data	OUT	flag indicating data on sbus
sbus(15:0)	INOUT	bus for cell transfer
stype(3:0)	IN	flag indicating type of incoming cell on sbus
sdata	IN	flag indicating incoming cell on sbus
sm_bus(15:0)	OUT	data bus for management cells
sm_address	IN	address of port granted SM bus
sm_data	OUT	flag indicating data on SM bus
sm_grant	IN	grant flag for SM bus
sm_request	OUT	request flag for SM bus

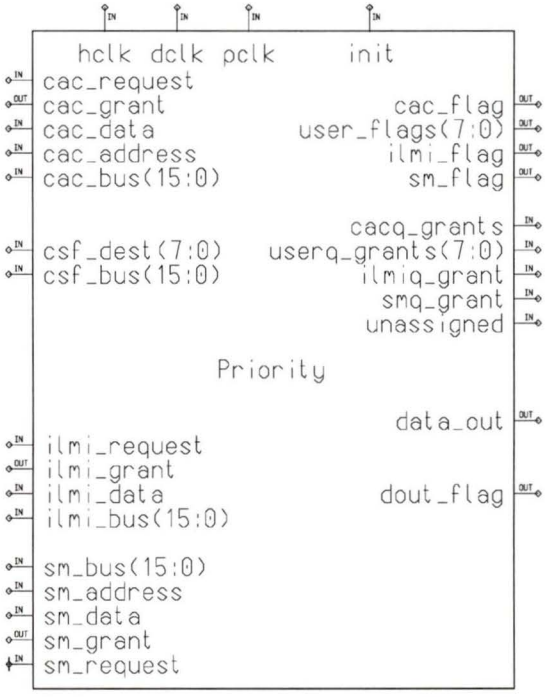
A.8.9 ILMI



PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
sbus(15:0)	IN	data bus for cell transfer to SIM outputs
stype(3:0)	IN	flag indicating cell type on sbus
sdata	IN	flag indicating data on sbus
ilmi_request	OUT	request flag for transfer of ILMI cell
ilmi_grant	IN	grant flag for transfer of ILMI cell
ilmi_data	OUT	flag indicating data on ILMI bus
ilmi_bus(15:0)	OUT	data bus for transfer of ILMI cells

A.9 Sub Output Module Components

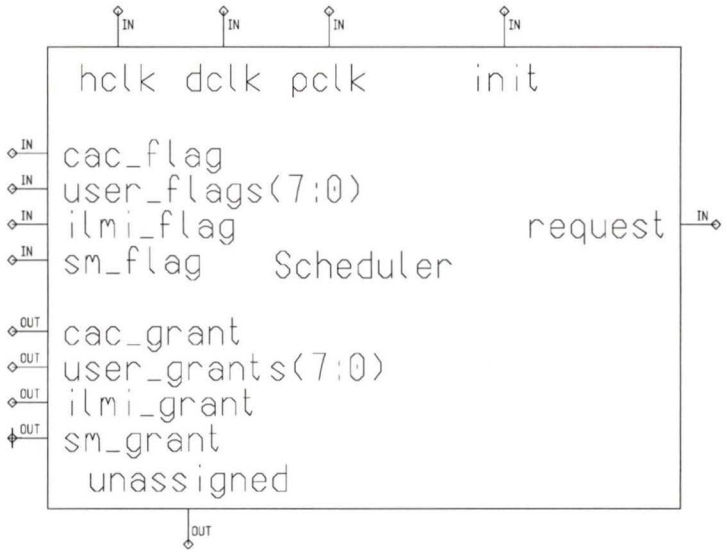
A.9.1 Priority



PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
cac_request	IN	request flag from CAC for cell transfer
cac_grant	OUT	grant flag to CAC for cell transfer
cac_data	IN	flag indicating data on CAC bus for SOM
cac_address	IN	port address for data on CAC bus
cac_bus(15:0)	IN	data bus for signalling cells
csf_dest(7:0)	IN	destination indication
csf_bus(15:0)	IN	cell data bus
ilmi_request	IN	request flag to transfer ILMI cell
ilmi_grant	OUT	grant flag for ILMI cell transfer
ilmi_data	IN	flag indicating data on ILMI bus
ilmi_bus(15:0)	IN	data bus for ILMI cell
sm_bus(15:0)	IN	data bus for managment cells
sm_address	IN	port address for data on SM bus
sm_data	IN	flag indicating data on SM bus for SOM
sm_grant	OUT	grant flag to SM for cell transfer

PIN	DIRECTION	COMMENTS
sm_request	IN	request flag from SM for cell transfer
cac_flag	OUT	flag indicating occupancy of CAC output buffer
user_flags(7:0)	OUT	flags indicating ocupancy of user priority output buffers
ilim_flag	OUT	flag indicating occupancy of ILMI output buffer
sm_flag	OUT	flag indicating occupancy of SM output buffer
cacq_grant	IN	flag granting write access to CAC output buffer
userq_grants(7:0)	IN	flags granting write access to user priority classes
ilmiq_grant	IN	flag granting write access to ILMI output buffer
smq_grant	IN	flag granting write access to SM output buffer
unassigned	IN	flag requesting generation of unassigned cell
data_out	OUT	serial data line for cell writing
dout_flag	OUT	flag indicating cell departure

A.9.2 Scheduler



PIN	DIRECTION	COMMENTS
hclk, dclk, pclk	IN	clock signals
init	IN	initialization signal
cac_flags	IN	flag indicating occupancy of CAC output buffer
user_flags(7:0)	IN	flags indicating ocupancy of user priority output buffers
ilmi_flag	IN	flag indicating occupancy of ILMI output buffer
sm_flag	IN	flag indicating occupancy of SM output buffer
cac_grant	OUT	flag granting write access to CAC output buffer
user_grants(7:0)	OUT	flags granting write access to user priority classes
ilmi_grant	OUT	flag granting write access to ILMI output buffer
sm_grant	OUT	flag granting write access to SM output buffer
unassigned	OUT	flag requesting generation of unassigned cell
request	IN	cell request flag

Appendix B

Simulation Methods

The compiled VHDL code for this ATM switch implementation was simulated using the Quicksim package found within the Mentor Graphics environment. The basis for simulation is the scheduling of events on the input ports of a components, and observing the output ports. The stimulus for these events was provided in the form of force files. These files list the input ports, the input values, and the time at which events on the ports are scheduled.

Force files were developed for the arrival of cells of different type and priority. For one port, the arrival of a series of cells was simulated to study cell stream separation , input queuing, and header generation. Additionally, the arrival of cells at multiple ports, and the subsequent arbitration of switch resources, was simulated. Each component was also simulated indiviually to verify its performance, before the higher layer architectures were simulated.

The timing diagrams of Chapter 5 were generated through tracing the port values over time. These traces were created for the full switch as well as for the SIM, SOM and CAC-IM arbitration.

VITA

James Andrew Gilderson
Born March 22, 1970, Ajax, Ontario

Educational Institutions Attended:

University of Victoria	1994 to 1995
McMaster University	1989 to 1993

Degrees Awarded:

Bachelor of Engineering	McMaster University, 1993

Honours and Awards:

B.Eng. studies

Canada Scholar

Chancellor's Scholar

NSERC Summer Undergraduate Research Assistant

Publications:

Conference

J. Gilderson, F. El-Guibaly and V. K. Bhargava, "VHDL design of an ATM switch", *Proceedings of IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing*, pp. 100-103, Victoria, British Columbia, Canada, May 17-19, 1995.

Technical Reports

J. Gilderson, F. El-Guibaly and V. K. Bhargava, "*VHDL ATM Switch Design*", Technical Report VMC/2, Dept. of Electrical and Computer Engineering, University of Victoria, 1995.

Partial Copyright License

I hereby grant the right to lend my thesis (or dissertation) to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis/Dissertation: **VHDL Design of an ATM Switch**

Author [redacted]
(Signature)

James Andrew Gilderson

Dec. 20 / 1995
(Date)