A RISC-based ATM Network Interface: processing, architecture, scalability and performance

by

Mohamed Elbeshti

B. Sc. (Computer Science)

Bright Star University, 1987, Libya

Thesis

submitted in partial fulfillment of the requirements for

the Degree of Master of Science (Computer Science)

Acadia University

Spring Convocation, 2000

0-612-51995-3

Table Of contents

List of Figures

List of Tables

Dedication

To the memories of my family, specially to my Mother

Acknowledgments

The decision to write a thesis represented a commitment to achieving academic enrichment and personal development. The completion of this thesis signified a milestone in achievement, of which I am proud. This would not have been possible without the help of some others.

I would like to extend my heartfelt gratitude to Dr. Ali Elkateeb for his assiduous dedication to me amidst his other pressing commitments. His steadfast faith and spirit motivated me to achieve nothing but the highest standard of excellence. This thesis would not have been possible without Elkateeb's support and encouragement. Thanks also extend to Dr. E. Eberbach for being my internal examiner and Dr. Hafizur Rahman for being my external examiner.

A special thanks to the Libyan Educational Secretaries (Tripoli-Libya) for their sponsorship.

Finally, I need to pay a special tribute to my family and friends. Also to Dr. Peter Smale and Carol Luczak for helping. My parents have always believed in me, and without them, this work would not have been possible.

Abstract

The question of whether to design the processing core of a network interface (NI) using a custom made hardware or an embedded processor for ATM Segmentation and Reassembly function is certainly an important one that has been addressed by many NI researches. The embedded processor core can be very useful in providing the following important features to network interfaces: simplicity, shorter developing cycle time, low cost, and flexibility to support protocol changes and perhaps new protocols. However, it is not clear what the scalability of NIs would be if their designs were based on embedded RISC core to support different high-speed transmission lines.

This work investigates the use of the Embedded RISC core in the ATM NI design. A cycle accurate VHDL-based simulator has been developed to measure the amount of processing required for ATM network interface design that support different transmission line speeds. The results have shown that a simple and cost effective embedded RISC core running under 85MHz can be used as a processing element in a high-speed ATM network interface. This core can support a wide range of transmission line speeds, up to 1.2Gb/s and 2.4Gb/s, for Reassembly and Segmentation functions respectively. We believe that this research can also be used as a guidance work for the ATM NI design.

List of Definitions of Abbreviations

AAL	ATM Adaptation Layer
ATM	Asynchronous Transfer Mode
B-ISDN	Broadband Integrated Services Digital Network
BOM	Beginning Of Message
CAM	Content Addressable Memory
CB	Circulation Buffer
COM	Continuation Of Message
CPCS	Common Part Convergence Sublayer
CRB	Cell Reassembly Buffer
CS	Convergence Sublayer
CSB	Cell Segmentation Buffer
DMA	Direct Memory Access
EOM	End Of Message
HI	Host Interface
HNIC	Host-Network Interface Communication
IEEE	Institute of Electrical and Electronic Engineers
ITU-T	International Telecommunication Union-Telecommunication Sector
LI	Line interface
MID	Message Identifier
MIPS	Million Instructions per Second

NI	Network Interface
RBI	Receiver Buffer Interface
REP	Reassembly Embedded Processor
RISC	Reduce Instruction Set Computer
SAR	Segmentation And Reassembly
SBI	Sending Buffer Interface
SEP	Segmentation Embedded Processor
SONET	Synchronous Optical Network
SPIM	MIPS spelt back work
SSM	Single Segment Message
VC	Virtual Circuit
VCI	Virtual Channel Identifier
VHDL	Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (HDL)
VPI	Virtual Path Identifier

Chapter 1

Introduction

In the past few years communication networks have been advancing rapidly in providing new services, improving their bandwidth and integrating new technology. Also, the network interfaces have been improved considerably. Such interfaces, capable of providing above Gbps speed, have been researched for different network protocols such as ATM, fibre channel and Gigabit Ethernet [LPete96, Desi97 and CGeor97]. Improvements to the network interfaces have led to support a new generation of applications for videoconferencing, video telephone, multimedia, etc.

As the speed of the networks have exceeded the Gbps, the design and implementation of high-performance Network Interfaces (NI) have become very challenging. One of the main challenges is the processing core design that is required for network interface protocols.

The approach of partitioning the processing of the protocol by allowing some functions to be processed on the NI and leaving the others for the host processor has reduced the amount of processing that the host processor usually does if NI is not used. As an example, the lower-level of the ATM protocol such as Segmentation and Reassembly (SAR) are processed on NI and the higher level protocols are left to be processed by the host. This approach was proposed and used in much researches

[ECoop91, DBru93, STraw93, CKim98, and RHosb99]. Other approach allows the NI to perform all the protocol processing without the host processing involvement [ZDitt97].

Generally there are three possible methods that may be used to process the network interface protocols:

1. General-purpose embedded processor [DBru93].
2. Fully customised logic [STraw93].
3. Programmable VLSI engines [CGeor97, CKim98, and RHosb99].

The general-purpose embedded processors may not provide the same level of performance as the other method offered, but they have better flexibility and they could easily accommodate protocol revision or even a new protocol. The wide availability of these processors has contributed to the low developing cost for network interfaces. Using these processors in designing the network interface makes the data path very simple and, hence makes their design simple too. An example of such network interface design is the one developed at Bellcore that supports 622 Mbit/sec ATM Network Interface for DEC TURBO channel which uses a pair of 33MHz Intel960 processor [DBru93].

In this thesis, we study and investigate the design of an ATM Network Interfaces (ANI) based on the use of the specialized embedded processor of a Reduced Instruction Set Computer (RISC) core type. Also, we have analyzed the amount of processing required by the ATM interfaces for both outgoing and incoming messages.

The following steps have been taken in our work:

1. We simulate an ANI model to support both ATM adaptation layer3/4 and AAL5 using SPIM S20 simulator to process the ATM Segmentation and Reassembly (SAR) protocols [Patt98]. This simulation is used to measure the amount of processing required for ATM protocols and data movement operations
2. We also simulated the ANI by using a high speed integrated circuit (VHSIC) hardware description language (HDL) VHDL. The IEEE *1164-1993 standard* running over the Xilinx foundation version 1.5 was used for our simulation to test our architectural model [Xili98, Xilinx99].
3. The RISC clock rate was measured for Segmentation and Reassembly protocol processing supporting the AAL3/4 and AAL5.
4. The RISC architecture was investigated to includes the appropriate instruction set, pipeline stage length and techniques to eliminate data and branch hazard.
5. We used DMA to assist the data movement activities, while the RISC was free to do other required processing. The speed of the DMA was investigated.

This thesis is divided into 6 chapters. Chapter 2 gives a general description of the NI design. The chapter also includes the protocol architecture of ATM network, ATM cell format and the concept of ATM Adaptation Layer (AAL). Finally, the chapter concluded with overview of related work. Chapter 3 shows the SPIM simulator model for ATM network and the SPIM simulator results. Chapter 4 described the VHDL Model architecture for ATM NI and the results for both AAL3/4 and AAL5. The design issues related to RISC core that specifically implemented for high-speed ATM host-network interface applications such as the instruction types at the RISC core and the pipeline stage are investigated in chapter 5. Finally the conclusion and the future works is discussed in chapter 6.

CHAPTER 2

Overview of ATM Network Interface

2.1 Introduction

In every workstation, the Network Interface (NI) is usually connected to the workstation's I/O bus and delivers messages to the host. The NI also receives messages from the host and then delivers them to the other end over the transmission line (Figure 2.1).

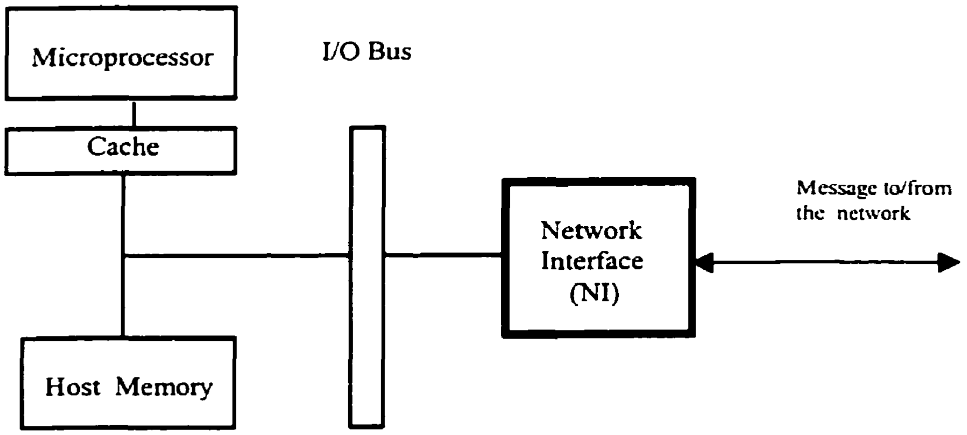


Figure 2.1: Workstation architecture

The network interfaces usually have two parts. The first part is the Line Interface which connects the workstation to the network line. The second part is the Bus Interface which connects the NI to the host. The Bus interface serves as a buffer between the NI and the host for receiving and transmitting messages.

The NI is generally designed to work for a specific network. The complexity of the NI design is basically dependent on transmission line speed and the protocol functions processing. In cases where the transmission line is running at moderate speed and the functions that are required to be processed by the NI are primitive, the NI can be very simple and does not need an interface-based processing because most of the processing can be done on the host (Figure 2.2).

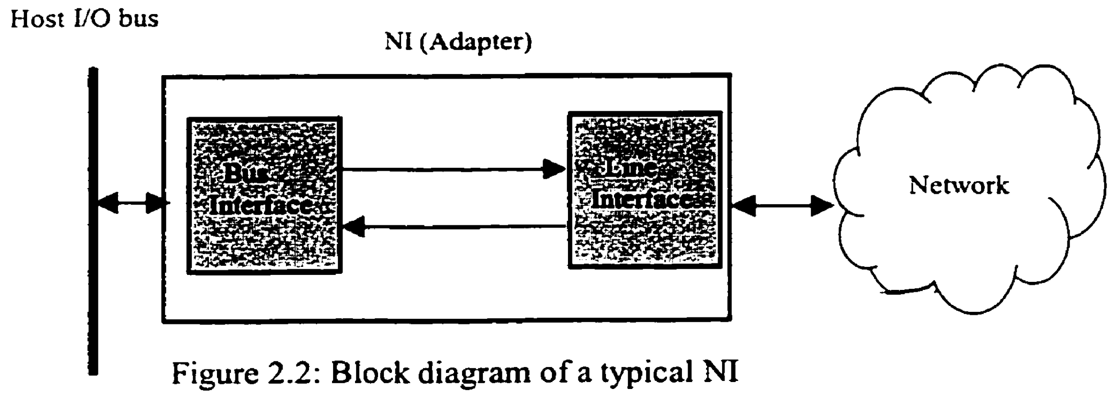


Figure 2.2: Block diagram of a typical NI

In other cases, where the transmission line is running at high speed and the functions processed by the host are large, performing all processing by the host will reduce the host capability of performing its normal job. Thus, an interface-based processing capability that removes the burden from the host processing becomes very important (Figure 2.3).

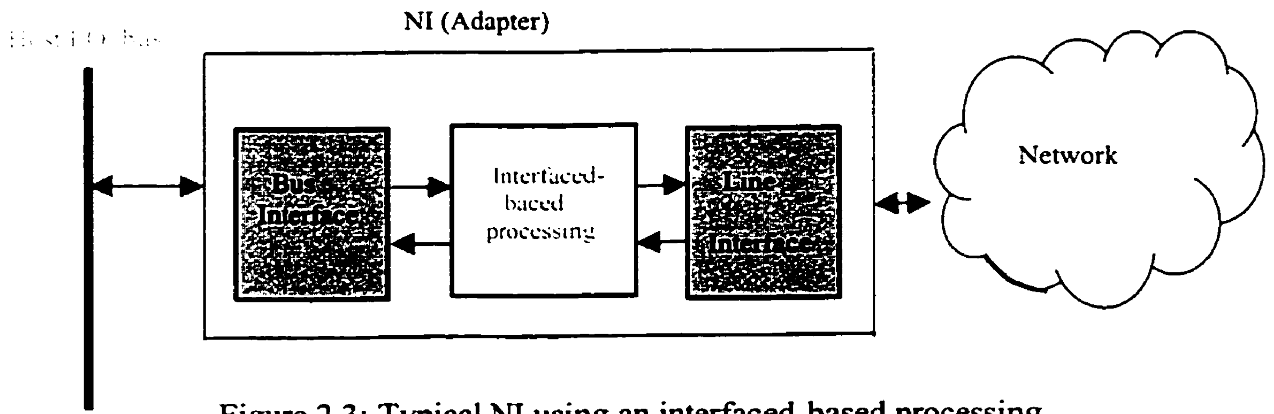


Figure 2.3: Typical NI using an interfaced-based processing

As the focus of this research is on the ATM network interfaces design, we believe that it is important to start by reviewing the ATM standard. Such a review will familiarize the reader with the ATM terms, concepts, and architecture. At the end of this chapter, a literature survey and related works in section 2.7 will be presented.

2.2 The Basic principles of ATM

The Asynchronous Transfer Mode (ATM) was born out of a standardization effort for Broadband ISDN (B-ISDN) which began in the CCITT in the mid 1980s. In the early 1990s, the data communications community saw the ATM standard as a promising candidate for networking in the local area. It was seen as a scalable method for the provision of high-speed network connections to routers and hosts. Currently ITU-T has steadily continued its work with respect to standardization of ATM, filling in the details related to operations and the transmission/reception a block of user data, and traffic characterization parameters.

There are many reasons why ATM is important for current and future networks. Firstly ATM can meet the bandwidth demands by offering a scalable range of transmission rates, such as T1/DS-1 (1.5 Mb/s), T3/DS-3 (44.7 Mb/s), OC-1 (51Mbps), OC-3 (155 Mb/s), OC-12 (622 Mb/s), OC-24 (1.244 Gbps), OC-48 (2.4 Gbps) [ITU-93, LPete96]. The standard for OC-192 (10Gbps) is already under development [CGeor97, LPete96]. Secondly there is the need for a single universal network which must meet all the user's requirements such as moving data, voice and video over a single network. Thirdly, ATM allows multiple logical connections to be multiplexed over a single physical link. For these proceeding and many other reasons, ATM will continue to be a popular networking technology despite the rapid progress in other network technologies such as Gigabit Ethernet.

2.3 ATM Cell

ATM uses a 53-byte cell [ITUR93] to transport data. The 5 byte header is primarily used for the association of cells to virtual connections and traffic management while the 48-byte payload of cells are carried transparently from the source to the destination.

There are a number of advantages to using fixed size packets in a communication network rather than the more traditional approach of using variable length packets. First, each cell will have a small amount of queuing delay which is useful for higher priority cells to meet the high ATM speed rate at the switch or the end node [LPete96, Comp98]. Second, packet lengths do not need to be calculated and the header does not need to carry length information. Third, it is simple to discover the delineation of cells with the fixed size cell.

2.3.1 ATM Cell format

ATM cell comes in two different structure formats, user-network interface (UNI) and network-network interface (NNI). The UNI cell format is used when the transmission cells are between user and network. The NNI cell format is used when transmission cells are between switches.

The header includes information about the contents of the payload and the method of transmission. The sections in the header are a series of bits that are recognized and processed by the ATM layer, except the CRC that is processed by the physical layer [Tray93, DBru93, RHosb99].

Sections included in the header are: Generic Flow Control (GFC) which appears only at UNI while it is added to the VPI at NNI, Virtual Path Identifier (VPI), the Virtual Channel Identifier (VCI), Cell Loss Priority (CLP), Payload Type (PT), and Header Error Control (HEC). The payload portion of the ATM cell contains the data to be transmitted. Figure 2.4 shows the ATM cell structure.

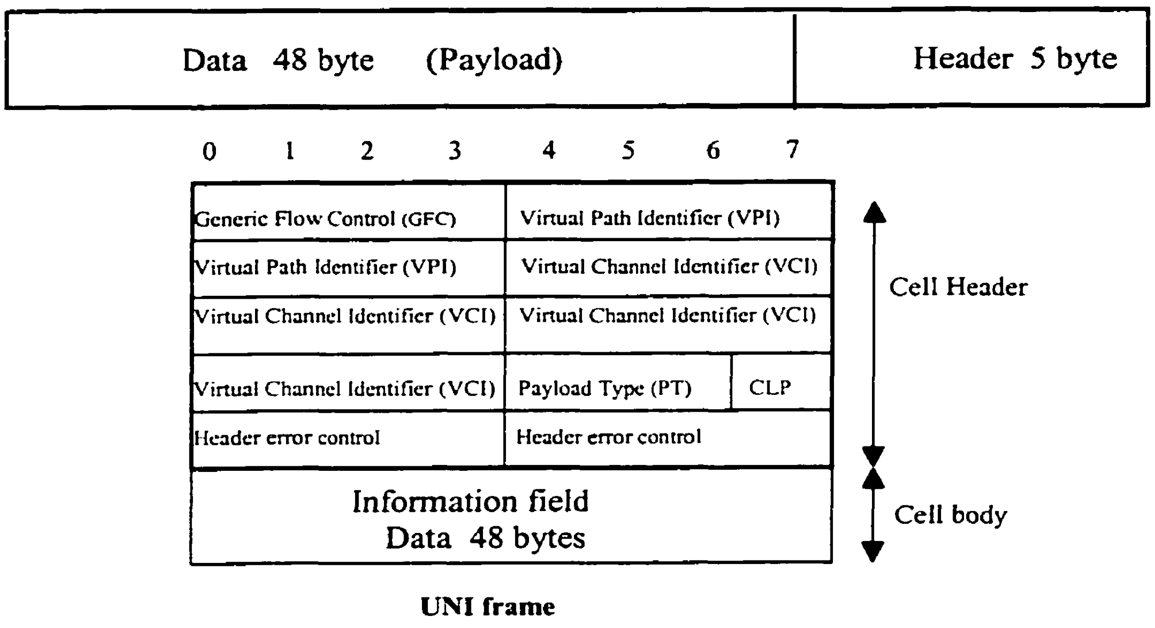

Data 48 byte (Payload)
Header 5 byte
0 1 2 3 4 5 6 7
Generic Flow Control (GFC)
Virtual Path Identifier (VPI)
Virtual Path Identifier (VPI)
Virtual Channel Identifier (VCI)
Virtual Channel Identifier (VCI)
Virtual Channel Identifier (VCI)
Virtual Channel Identifier (VCI)
Payload Type (PT)
CLP
Header error control
Header error control
Information field
Data 48 bytes
Cell Header
Cell body
UNI frame

Figure 2.4: ATM Cell

2.3.1.1 Header Description

The first four bits in the header for the UNI cell format include GFC presented as bits in the ATM header to support flow control. This mechanism was proposed by the ITU-T recommendation [ITUR93]. The VPI and VCI provide information on the path that the cell will take during its transmission. The PT section contains three bits that indicate whether the payload contains user data or layer management information. User data is data of any traffic type that has been packaged into an ATM cell. An example of management information is involved in call set-up. This field also notes whether the cell experienced congestion.

Payload Type	Field definition
000	User data cell,AAU=0 congestion not experienced
001	User data cell,AAU=1 congestion not experienced
010	User data cell.AAU=0 congestion experienced
011	User data cell,AAU=1 congestion experienced
100	OAM F5 segment associated cell
101	OMA F5 end-to-end associate cell
110	resource management cell
111	reserved for future function

Table 2.1: Payload Type (PT) description

The CLP bit indicates the loss priority of an individual cell. Cells are assigned a value of 1 or 0 to indicate that they are either high or low priority. A cell loss priority value of zero indicates that the cell contents are of high priority. A cell that has value 1 in its CLP is discarded if congestion occurs in the network. Cells with a high priority will only be discarded after all low priority cells have been discarded. The last part of the ATM header is an 8-bit header error control field that consists of error checking bits. This field provides error checking only for the header field, not for the payload.

2.4 Virtual Connection

The ATM network service is connection-oriented. This means that a connection must be set up between two ATM hosts before user data can be transmitted. In ATM terminology, the connection set-up is called signaling. Once two users accept the connection, then the virtual connection is dedicated to the source and the destination. ATM can operate one or more virtual connections over a single physical link. A Virtual Channel (VC) is used to describe the unidirectional transport of ATM cells associated by a common identifier value specified by a 24-bit VC which is assigned at call set-up [ITUT95]. This common identifier is the VCI/VPI contained in the ATM cell header part of each cell. The VPI is fixed to 8 or 12 bits long and supports 256 or 4096 virtual paths. Each path can be composed of up to 64K Virtual Channel Identifier by its VCI [Marti95]. A VCI value is used to distinguish VCs of a VP where these VCs allocate at the end ATM point as well as within the network. Switches containing a routing table of switch ports and connection identifiers are used to interconnect ATM hosts and networks. Each cell is transported through the switch based on the connection identifier in the cell's header.

It should be noted that two VCs belonging to two different VPs could share the same VCI value. Thus, VCI values are only significant within VPs. This concept is useful when two users need to set-up a number of separate connections to each other. In addition, VCs demanding similar quality of services from the network can be multiplexed together.

For example, video telephone could be sent over the network. It would be divided into three components: one VCI for voice, one for video and one for data (Figure 2.5).

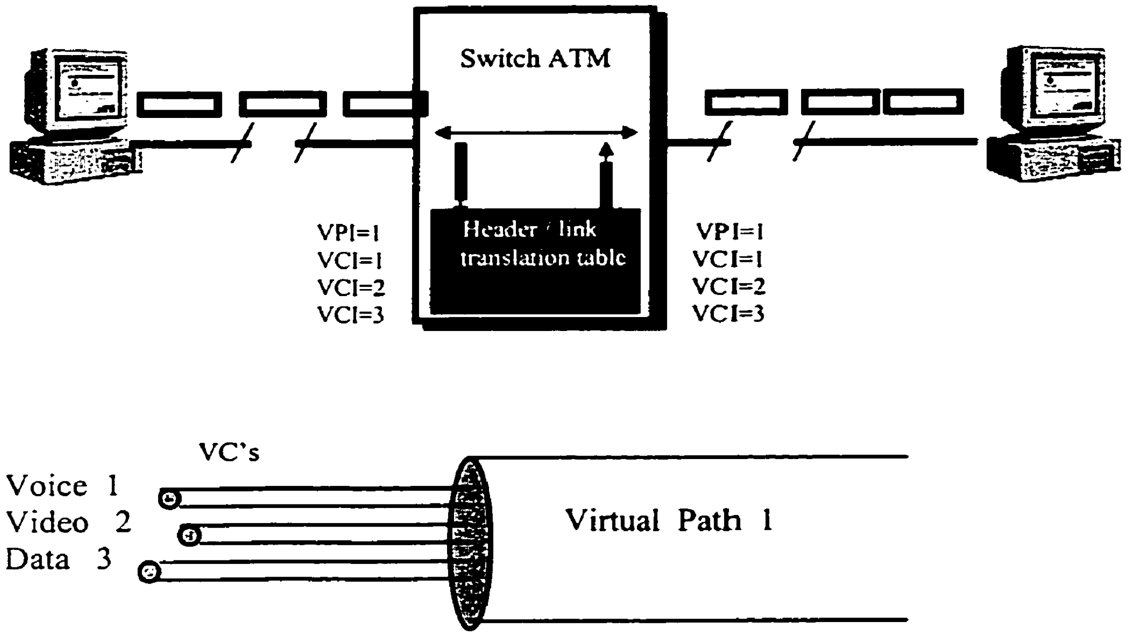


Figure 2.5: VCI and VPI Connections in ATM

2.5 ATM Protocol Architecture

The basic protocol architecture for a B-ISDN model between user and network is issued by the ITU-T, which is composed of three separate plans and four layers (Figure 2.6). The physical layer of the ATM protocol is divided into two sub-layers, the Transmission Convergence (TC) sub-layer and the Physical Medium (PM) sub-layer. The TC is responsible for the generation and verification of the header error control byte, checking idle cells, and cell delineation.

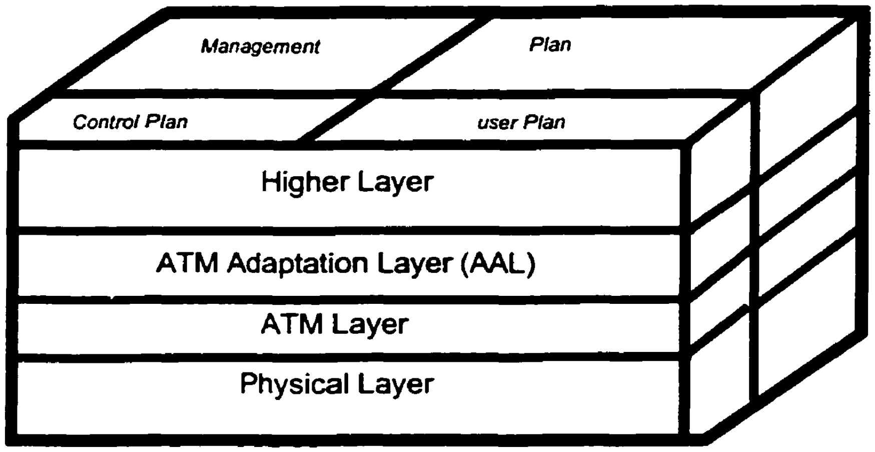


Figure 2.6: ATM Protocol Architecture

The PM is concerned with converting the signal into electrical or optical output for the transmission of data over a transmission media at different data rates. The ATM layer operates independently of both the underlying physical layer and the AAL layer above it. The ATM is responsible for a number of functions involving the contents of the cell header. In the transmit messages from source, the Segmentation and Reassembly-Protocol Data Unit (SAR-PDU) is accepted from the AAL and encapsulated in ATM cell payloads where the ATM layer generates various cell headers including VPI and VCI fields. On the receiving side, the cell headers are extracted from their payload and the payload part is passed to the AAL layer. Cell payloads are not manipulated at the ATM layer. Other functions performed by the ATM layer are multiplexing and demultiplexing of cells of different VC into a single cell stream on a physical layer.

The AAL of the protocol reference model accepts variable length PDUs from the higher layer protocol and maps these into fixed size ATM cell payloads. However, different services require different AALs. The AAL layer is further sub-divided into two sub-layers: the Convergence sub-layer (CS) and the Segmentation and Reassembly

(SAR) sub-layer. The CS provides services which include the multiplexing of higher layer messages and cell loss detection/recovery. The SAR sub-layer accepts the CS's frame and segments it into ATM cell payloads. Then the SAR sub-layer executes the inverse operation of resembling the cells of a VC into data units to be delivered to the higher layer. The CS is further sub-divided into a Common Part CS (CPCS) and a Service Specific CS (SSCS). The function of the former is dependent upon the higher layer services that are using the AAL. The CPCS performs functions such as padding and adding headers and trailers to the entire AAL frame before passing to the underling SAR sub-layer. The SSCS may operate over the CPCS.

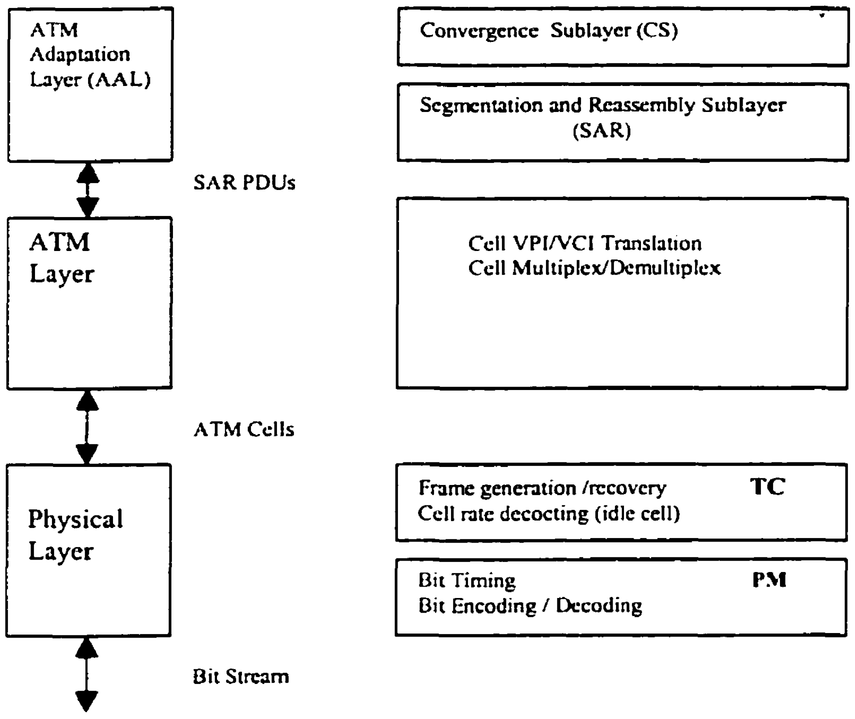

ATM Adaptation Layer (AAL)
Convergence Sublayer (CS)
Segmentation and Reassembly Sublayer (SAR)
SAR PDUs
ATM Layer
Cell VPI/VCI Translation
Cell Multiplex/Demultiplex
ATM Cells
Physical Layer
Frame generation /recovery
Cell rate decocting (idle cell)
TC
Bit Timing
Bit Encoding / Decoding
PM
Bit Stream

Figure 2.7 : Layers Description

AAL maps the user management PDU into small blocks to fit in the ATM cell of virtual connection, and vice versa.

2.5.1 ITUT-T I.363 list for AAL service

The AAL services supported by I.362, which represented B-ISDN ATM Adaptation Layer (AAL) specification [ITUT93], standard are:

- Handling of transmission error.
- Segmentation and Reassembly. End ATM nodes can transmit a large amount of data by breaking the data into small pieces to be fit in the small fixed cell and reassemble them at the destination.
- Handling of lost data condition.
- Adding some fields to the ATM payload to allow the processor at the end node to discover any cells missing from its sequence data.

2.5.1.1 AAL Classes

The services transported over ATM layers are classified into four classes shown in the Table 2.2. Each of these classes has its own specific requirements for the AAL. The services are classified in three basic parameters for these four classes .

1- Time relation between source and destination.

2- Bit rate (variable or constant bit rate).

3- Connection mode (connection or connection-less).

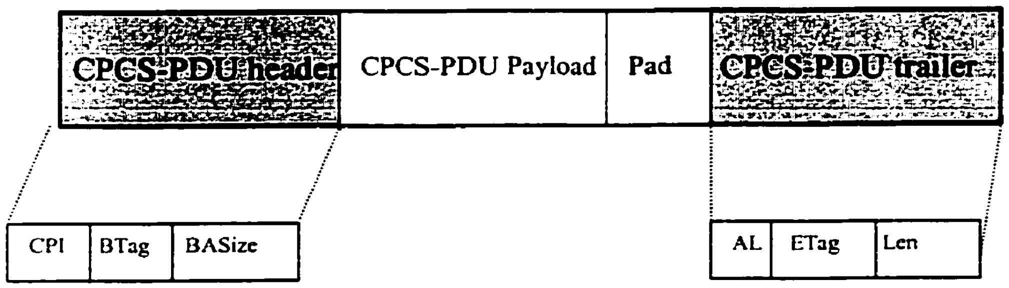
ITU-T defined one protocol type for each class of services named Type1 through Type 5 and these are known as AAL1, AAL2, AAL3/4, and AAL5.

	Description	Bit-rate	Type	Class
AAL1	Support connection-oriented services require information to be transferred between source and destination at a constant bit rate. (fixed bit rate video)	Constant	Connection Oriented	Class A
AAL2	Support connection-oriented services that do not require constant bit rates, but have timing and delay requirements. (Compressed video or sound)	Variable	Connection Oriented	Class B
AAL3/4	Is intended for both connection-less and connection-oriented services with variable bit rate	Variable	Connection Oriented and Connection-less	C/D
AAL5	Supports connection-oriented services that require variable bit rates. Delay and timing are not crucial. AAL5 is a simpler than AAL3/4, at the expense of error correction and automatic retransmission, but pays off with less bandwidth overhead and reduced implementation complexity.	Variable	Connection Oriented	C/D

Table 2.2: Traffic Classes and Criteria

2.5.2 ATM adaptation layer 3 / 4

The main function of AAL3/4 is to allow bigger size messages, where the length of these messages do not exceed 64 Kbytes, to be transported across the ATM network as a series of fixed length ATM cells. The user information to be segmented involves two different formats. The first format is Common Part Convergence Sublayer-PDU (CPCS-PDU). Variable-length of CPCS-PDUs payload are encapsulated in the CPCS-PDU frame format (Figure 2.8).



Common Part Indicator	(CPI)	1 octet
Beginning Tag	(BTag)	1 octet
Buffer Allocation Size	(BASize)	2 octet
CPCS Payload		1-65,535 octets
Padding	(Pad)	0-3 octets
Alignment	(AL)	1 octet
End Tag	(ETag)	1 octet
Length	(Len)	2 octet

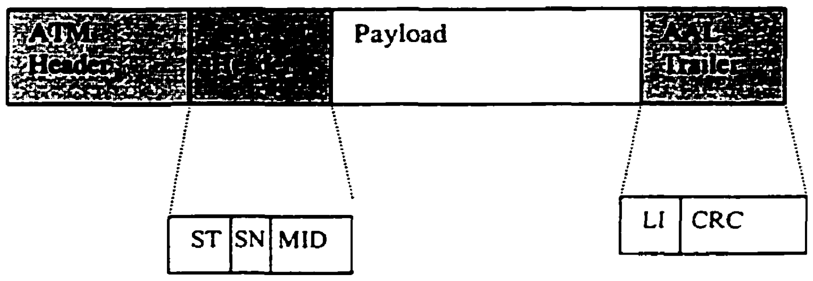
Figure 2.8: CPCS-PDU format for AAL3/4

2.5.2.1 Header Description of CS-PDU

The CPI is used to interpret the subsequent fields for the CPCS functions in the CPCS-PDU header and trailer. It indicates which version of the CS-PDU format is in use. Only the value '0' is currently being used [ITUT93]. The Beginning Tag (BTag) field and End Tag (ETag) fields allow the association of the first and the last SAR-PDUs of one CPCS- PDU. Since variable length PDUs are encapsulated, the length of the CPCS payload varies. These Tags also protect each PUD against the situation in which the loss of the last cell of the current PDU and of the first cell at the beginning of the next PDU to be joined as one PDU at the destination.

The buffer allocation size (BASize) is used to indicate, to the receiver side, the max buffer size required allocating the current CPCS-PDU. Padding (PAD) fields contain 0-3 octets, which are not part of the user information, positioned between the CPCS-PDU payload and the 32-bit aligned CPCS-PDU. The alignment field should be set to '0'. Length field has two purposes. The assignment of the length of the CPCS-PDU payload and to detect the loss or gain of information at the receiver side. The CPCS-PDU frame is passed to the SAR sub-layer where it is segmented into equal chunks.

The second stage format when the CPCS-PDU frame segments into small pieces. Each piece with 44-byte of CPCS-PDU plus 4 bytes of header and trailer is carried with each cell (Figure 2.9).



Segment Type	(ST)	2 bits
Sequence number	(SN)	4 bits
Multiplexing Identifier	(MID)	10 bits
Length Indicator	(LI)	6 bits
Cycle Redundancy Check	(CRC)	10 bits
SAR Payload		44 octets

Figure 2.9: SAR Structure for AAL3/4 cell format

2.5.2.2 Header Description of AAL3/4 frame

In the first field of the AAL3/4 is Segment Type (ST) which used to indicate the beginning of CPCS-PDU message (BOM), end of message (EOM), continuation of message (COM), or single segment message (SSM). See the Table 2.3.

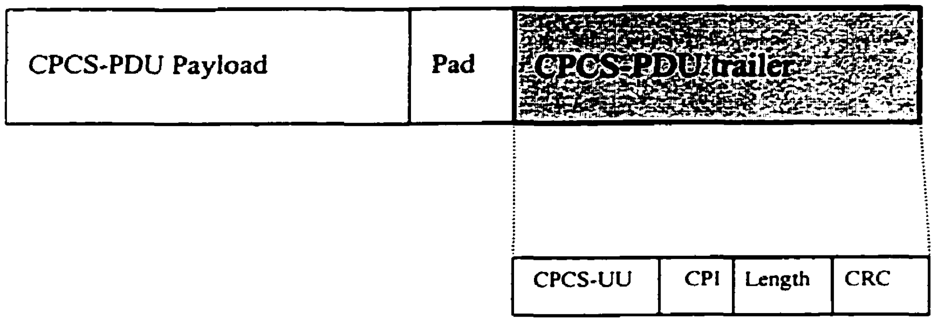
Value	Name	Meaning
10	BOM	Beginning of message
00	COM	Continuation of message
01	EOM	End of message
11	SSM	Single segment message

Table 2.3: AAL3/4 type field description

The next field is the sequence number (SN) which is used to recognize cell loss or a disordering cell. The multiplexing identifier (MID) field is used to identify SAR-PDUs belonging to particular SAR-SDU which can be assigned to help different PDU on a single connection. The two ends during the call set-up negotiate the range value of the MID field [ITUT95].

2.5.3 ATM Adaptation layer 5

In the AAL5, the frame of CS-PDU consists of the data portion, which is handed down by the higher-layer protocol, and the eight-byte trailer. At the CS Sub-layer, the AAL5 protocol does not specify any information for buffer allocation size, and CRC checking is fully performed on the entire message at the CPCS Layer.



CPCS Payload		> 65,535 octets
Padding	(PAD)	0..47 octets
CPCS User-to user indication	(UU)	1 octet
Common part identifier	(CPI)	1 octet
Length		2 octets
Cyclic Redundancy Check	(CRC)	4 octets

Figure 2.10: AAL5 CPCS-PDU frame format

The padding field (PAD) is located between the data and the trailer in the CPCS-PDU, where PAD size can vary from 0 - 47 octets, this ensures that the total size of a CPCS-PDU is a multiple of 48 octets of SAR data. The second field is the user to user, which contains one octet that is used to carry CPCS user information. The common part identifier (CPI) contains zeros in its field, indicating that the CPCS PDU contains user data. Other CPI values are for further study. The length field is used to indicate the CPCS-SDU payload's length in the CPCS-PDU. It is necessary to figure out the actual size of user data from its padding data.

The main feature missing in AAL 5 is multiplex identifier (MID) which has the ability to send multiple SAR connections on a single ATM layer connection cell over an active connection. Also, AAL5 uses the ST at ATM header to distinguish between the last or single SAR segment and the rest of the segments. AAL5 sets the ST value of '1' to identify the last cell of CPCS-PDU as a last cell of the CPCS-PDU frame. All the other cells such as BOM or COM will have value '0'. This procedure can eliminate the two bits ST field at AAL3/4 header.

2.6 Segmentation and Reassembly (SAR)

This mechanism allows the users to send a big message (less or equal 64 Kbytes) through the protocol layers to the ATM network where AAL has it is ability to cut the CS-PDU frame into small pieces as ATM payload. Furthermore, AAL can send these pieces to lower layer (ATM layer) which then completes the process by adding ATM header, and sending the cell to the physical layer, then to the transmission line. The Physical layer is responsible to add CRC at the fifth byte in the ATM header to help the receiver side accept corrected header information. This procedure is known as Segmentation. The receiving side accepts the packet from lower level (after extracting its header from the packet) and then reassembles the fragments back together at the destination. This is known as Reassembly. The general procedure is called Segmentation and Reassembly (SAR).

2.6.1 Segmentation and reassembly for AAL3/4

The AAL3/4 header and trailer which contain 2 octets are added to the 44 byte payload which is cut from the CPCS-PDU frame to be 48-byte and then sent it down to the ATM layer which will add its header to be a complete ATM cell format. In the receiving procedure the ATM layer will extract the ATM header and then pass the payload part to the AAL layer which extract the AAL header and trailer and pass the remaining payload to its related message format (Figure 2.11).

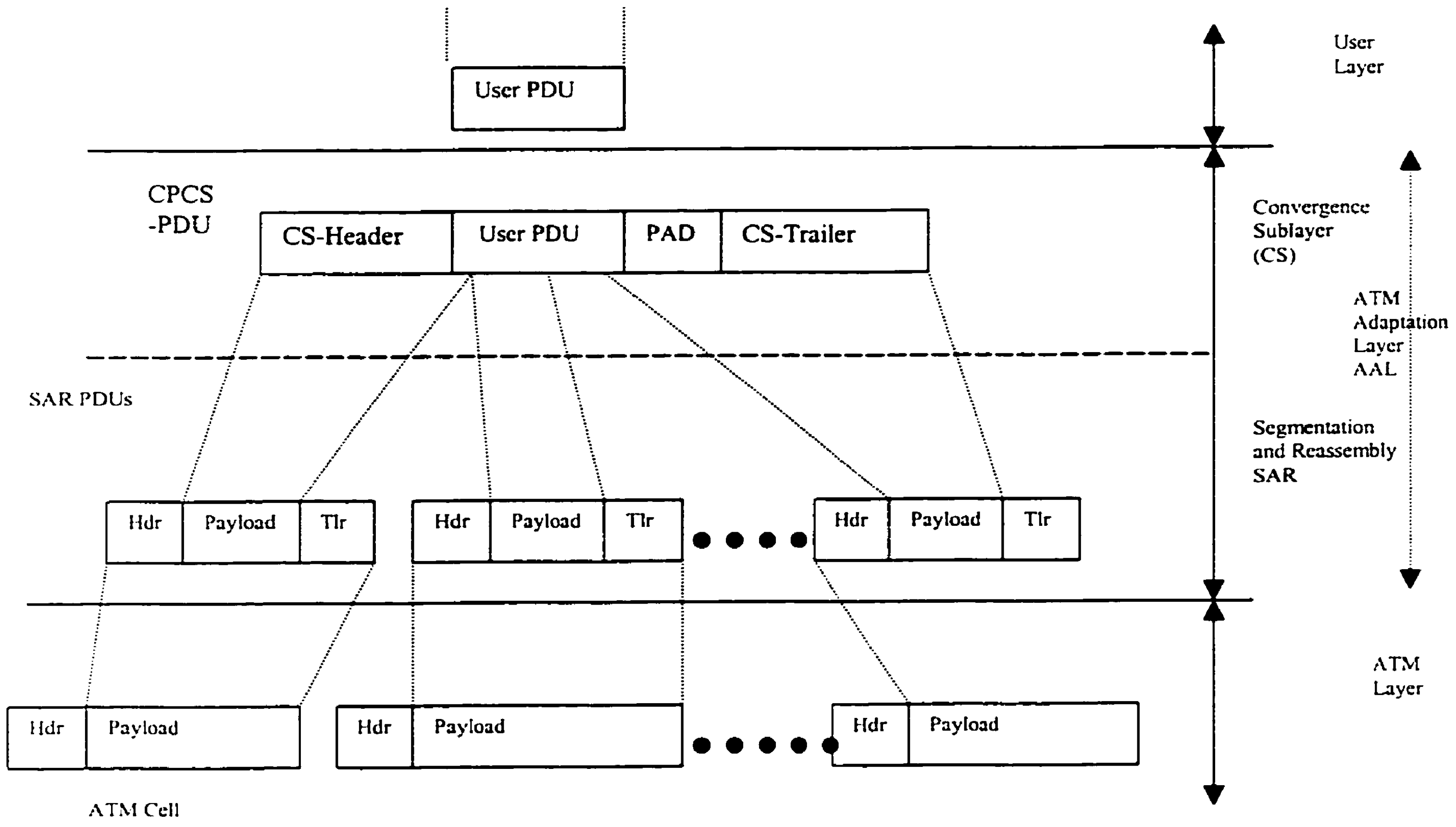


Figure 2.11: AAL3/4 Segmentation and Reassembly (SAR)

2.6.2 Segmentation and reassembly for AAL5

In the ATM layer there is less overhead than in AAL3/4 where the payload part can carry 48 bytes instead of 44 bytes. The improvement by decreasing the amount of processing at AAL layer makes AAL5 more efficient and attractive than AAL3/4 (Figure 2.12).

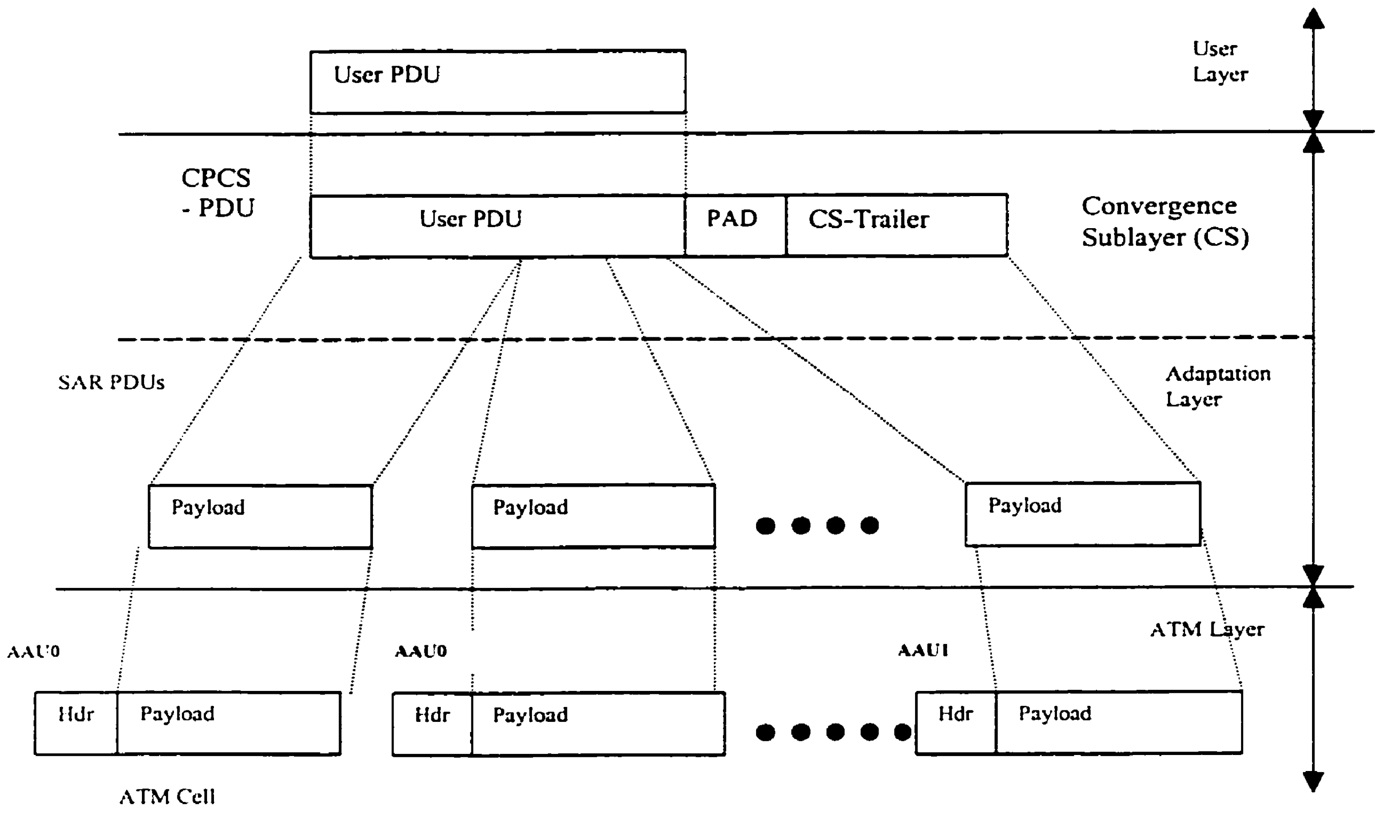


Figure 2.12: AAL5 Segmentation and Reassembly (SAR)

2.7 Different ATM interface architectures

It is very clear that using an embedded processor element on the NI will release the host processor from most, if not all, of the processing that is required by NI. A commonly asked question concerns how large an amount of processing may be done if the NI has no processing element. In the case where such processing is large, the cost of adding an embedded processor to the NI can be justified. Otherwise, the host can perform all the processing required by the NI functions. All the NIs that are used today have a certain processing element inside the NI to reduce or eliminate the host processing required for NI functions.

In this section we will demonstrate some of the ATM interfaces that have been designed to serve in ATM ends. Brief summaries about these important works are presented in this section.

a) The STS-3c interface has been developed by Traw and Smith at the University of Pennsylvania (UPenn) [STraw93]. They designed an interface for the R6000 workstation and does ATM segmentation and reassembly on the host network interface's memory. This interface is created with pure hardware to operate at STS-3c rate 155 Mbps to support AAL3/4 function. The protocol processing for SAR was divided into different components (such as VCI lookup controller, linked list manager, and segmentation controller). These components operate concurrently to give good performance for the Segmentation and Reassembly protocol processing, where data has been pipelined to process from one to another component.

   The on-board processor has been used to process the ATM protocol including the SAR functions. The Reassembly messages are performed in a local buffer to reduce interrupting the host CPU by each arrival cell. There is a necessity to change some of its component in order to run different protocols or new version of the same protocol.

b) An STS-12 622 Mbps ATM SONET stream interface has been developed by Davie [DBru93] at Bellcore. This interface was designed to work with a TURBO channel bus on a DEC station 5000 and processes the segment and reassembles messages using host memory. The operations include SAR protocol processing for AAL3/4 (also suitable for AAL5) done by two Intel 80960CA 33MHz microprocessors. These microprocessors give a great flexibility for deciding the transmitting and receiving of different VC to or from the ATM network, where each decision is done within 23 instructions before the next cell arrives.

Each side (receiver and transmitter) operates independently. However, they still need to communicate, in some cases, with each other in order to make the performance of receiving and transmitting data more efficient. DMA is used for data transfer.

This interface performed the processing needed for ATM and AAL layers by an on-board processor. It uses the host memory for reassembling the message and notifies the host after receiving a number of messages for a particular VCI (when memory buffer is full). This approach has a smooth design for the interface but it is not clear that if Intel 80960CA can sustain high-speed lines such as 1.2 Gbps and higher rate.

c) A 155 Mbps ATM host interface controller (ASIC) was designed in Electronic and Telecommunication Research Institute, Korea [CKim98]. This interface uses host memory to store the arrived messages. The ATM Subscriber Access Handler - Network Interface Controller (ASAH-NIC) is composed of a segmentation and reassembly engine to process SAR protocols related to AAL3/4 and AAL5. The segmentation engine requests the DMA to move a block of data ATM cell body from the host memory and store it in the temporary buffer (FIFO) in the network interface and then sent as a complete ATM cell to the transmission line. The reassembly engine requests the DMA to store the received cell body from interface's memory to the host memory. This request provides some information, such as the host memory address to which the data is stored, and the address of the local memory from which the cell body should be read and higher rate.

d) A 700 Mbps interface was designed by Richard and Wang at the British Columbia Advanced System Institute [RHosb99]. This interface uses a 32-bit embedded processor core and Receive Control Unit (RCU), and Receive Data Transfer Unit (RDTU) for the ATM, and SAR protocol processing for AAL5 reassembly. The reassembly memory subsystem is composed of pages that can be dynamically allocated for variable sized PDU's. The address at the beginning of the transfer sequence determines the memory to which the current page should be written. A special request is granted when the microprocessor determines that complete PDU has arrived in the local memory. The DMA mechanism is used for data transfer. More efficient data movement over 32-bit bus is enabled through 32-bit data paths.

Studying the previous ATM interfaces brought several issues to the foreground. The first issue is the distribution of protocol functions between the on-board processor in the network interface and the host processor. Processing of higher-level protocol functions should be performed by the host and the processing of ATM and AAL functions be performed in the network interface.

Second, most of the interface designers are focusing on eliminating the number of copying cells inside the NI, and also providing a fast device for data movement. Using the DMA in the area of the data movement is shown in the all previous works and also is addressed in [GPart94 and AElka00].

Third, the NI should reassemble the incoming cell to be a complete message (CPCS-PDU frame). One approach is to reassemble the message in the host memory, as shown 'b' and 'c,' or reassemble the message in the NI buffer, implemented in 'a' and 'd.'

Fourth, the architecture of NI should provide between the host and the network interface a flexible means of communication, which is useful in reducing the amount of interrupting time caused by the arrival cells. The NI should be designed not to interrupt the host for each arrival cell [GPart94]. The commands between the host and the NI can be passed through dual ported memory, as proposed in [Burc93], or through the FIFO queue buffer [Ckim98, RHosb99, AElka99 and Aelka00]. Finally, the adapter should be simple, scalable, small in size and low cost.

2.8 Conclusion

There is still little study on the impact of network interface design for gigabit networks, specifically in the area of the processing capability for multiple gigabit network interfaces. Such interfaces will require a high-speed processing unit to cope with the increasing speed of the transmission rate.

Most often, the use of general-purpose cores within the network interface design is very attractive due to their availability, short developing cost, and their simple NI design. However, such cores are not used in high-speed network interface designs because no clear indication whether such cores-based NI will be scalable, and because of that such core are not optimising for NI applications.

Chapter 3

ATM Network Interface Simulation

3.1 Introduction

Some of the challenges behind the building of ATM interfaces are: high-speed transmissions, ATM cell structures, and the ATM and AAL protocol processing. As the network speed increases, the time that is available to the NI to process the arrived cell will decrease, and therefore the processing unit inside the NI should be fast enough to finish the processing of one cell before the next one arrives. Since the ATM network uses cells which are very small (53-byte), the network interface in the transmission side should partition the original message into ATM cells. The whole message may require many cells to be sent out, one after the other, in order to transmit a complete message to the receiver. Fragmenting the original message into ATM cells is known as Segmentation function. At the receiver side, the original message will be constructed from these small cells. The reconstruction processing at the receiver is called Reassembly function. Segmentation and Reassembly (SAR) functions have put more challenges on the processing part of the ATM NI, especially where these cells belong to different messages of different applications. Other functions that NI should process are:

- The ATM header.
- The AAL header and trailer.

- Virtual Channel (VC).
- Data movement.
- Communication with host processor.

In order to understand and evaluate the processing which carried out by the NI, a simulation process will be required for NI model. Building a simulator for a model that has components used in real NI is a time consuming task; therefore, we decided to evaluate the processing by using a simple simulator. Our strategy was to estimate the NI functions processing requirement by using this simple simulator. Such estimation will help to decide whether the cost effective embedded RISC core is possible to be used in such applications.

The rest of this chapter describes our model and the simulation process. The chapter concludes with the results we obtained for both AAL3/4 and AAL 5 protocol processing.

3.2 The Simulation

In order to simulate the NI function in a very short period of time, we decided to use the SPIM S20 simulator [DPatt98]. It runs programs for the MIPS R2000/R3000 RISC microprocessors where it can read and immediately execute files containing assembly language. The simulator is a self-contained system for running these programs and contains a debugger and interface for a few operating system services [SPIM97].

The SPIM simulator is used in this work to process the ATM NI functions. Since in real ATM NI, the Segmentation and Reassembly functions are generally processed in two different processors and both are run in parallel, the simulation for the Segmentation function has been performed independently from the Reassembly function.

Two methods are possible for data movement and can be used in our simulation approach [LPete96, DPatt98]. The first method is programmed I/O (PI/O). In this method, the embedded processor takes the complete responsibility for moving the data portion from one place to another. The second method is to use DMA. The DMA will take responsibility for moving data from one place to another and eliminate the need for an embedded processor intervention to do that function. As the SPIM simulator does not have a DMA unit, we only let the embedded processor core simulate the initialization of the control information for the DMA controller and not the data movement itself. That has made the simulation processing very close to reality where the processor needs only to initialize DMA.

The simple simulator that we present in this chapter has used the R2000/R3000 to process the NI's functions and the simulator's memory to hold all NI buffers (Figure 3.1). These buffers are:

- The Line Interface (LI) buffer to store the arrival cells from the network line.
- The Host-NI communication (HNIC) buffer which is used to exchange control and status messages between the host and NI.
- The Host Interface (HI) buffer that is used for storing cells temporarily.
- The Circulation Buffer (CB) which is used to store all the address pointers for the free space that exists inside the HI buffer.

Clearly in real NI the LI, HI, and HNIC buffers are hardware components and we chose to place them in the simulator's memory. The Content Addressable Memory (CAM) that is required for holding the active identifiers and the support for the link-list mechanism, as will be described later, is also implemented inside the memory simulator. In the real NI, the CAM is implemented as a separate memory.

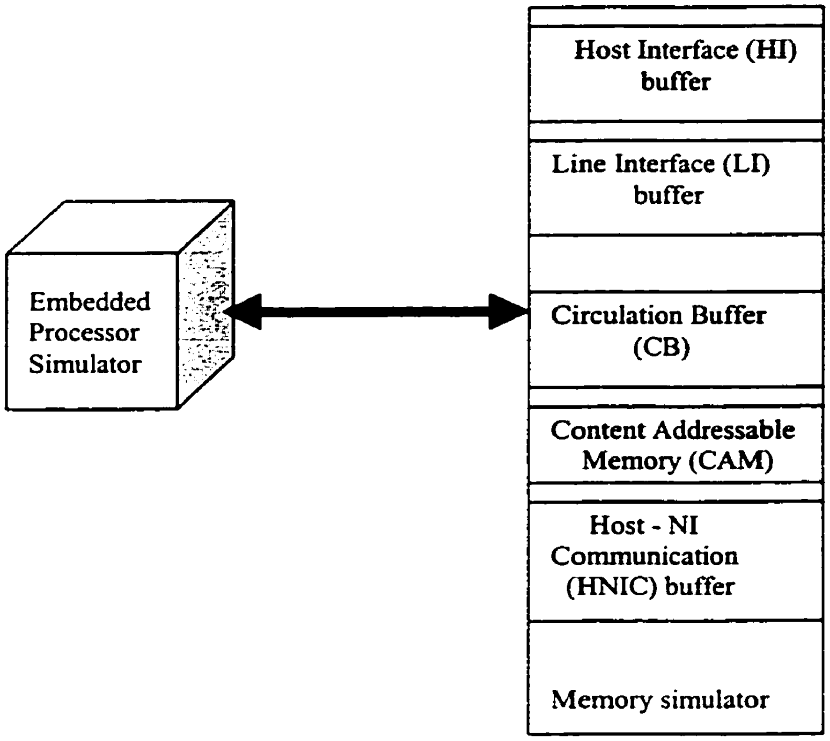

Host Interface (HI) buffer
Line Interface (LI) buffer
Embedded Processor Simulator
Circulation Buffer (CB)
Content Addressable Memory (CAM)
Host - NI Communication (HNIC) buffer
Memory simulator

Figure 3.1: Simple NI simulator structure.

All the above buffers are used for Reassembly functions processing where the Segmentation function processing requires only HI, LI, and HNIC buffers.

3.2.1 Buffers Description

3.2.1.1 Circulation Buffer (CB)

The ATM cells are fixed in length and the reassembly process is required to reconstruct the arrival cells in the HI buffer to a whole message and then deliver it to the host. The HI buffer is basically partitioned into smaller buffers, each of these buffers is equal to the size of an ATM cell. The CB is used to store all the pointers of there small buffers in order to monitor all the buffers within HI. The Reassembly Embedded Processor (REP) uses two 32-bit registers to control the CB. These registers hold the head-of-the-CB pointer and the tail-of-the-CB pointer. The head-of-the-CB pointer refers to the first available space on the CB that can be used to reassemble the incoming cells, where the tail-of-the-CB pointer is used to indicate the location of the last available space inside the CB (Figure 3.2).

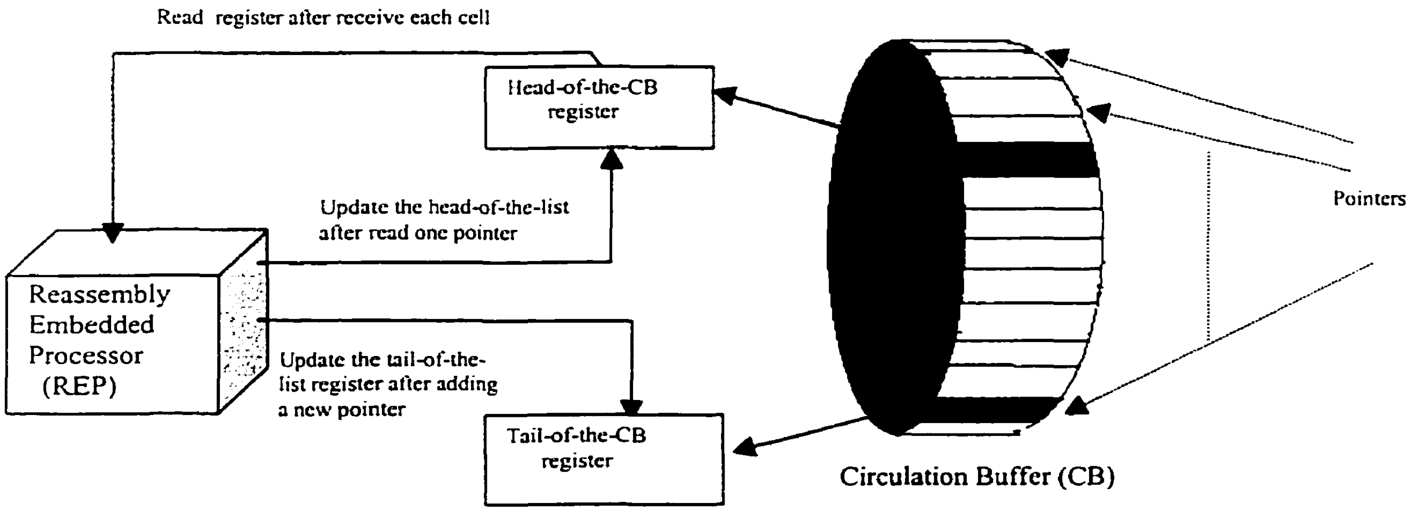

Read register after receive each cell
Head-of-the-CB register
Update the head-of-the-list after read one pointer
Reassembly Embedded Processor (REP)
Update the tail-of-the-list register after adding a new pointer
Tail-of-the-CB register
Pointers
Circulation Buffer (CB)

Figure 3.2: Circulation Buffer (CB) architecture.

The REP used for reassembly functions processing will update the head-of-the-CB register whenever a new cell arrives. Moreover, it updates the tail-of-the-CB register whenever the host reads the message from the HI.

3.2.1.2 Host-NI Communication (HNIC) buffer

The NI has to communicate with the host to organize sending and receiving messages and exchange control and status information. In the Reassembly unit, the HNIC is split into two sections to store the control and status information (Figure 3.3). The first section of HNIC is used to store information, which is sent by the REP for the host. The REP, after storing the complete CPCS-PDU frame inside the HI buffer, sends the Start-address and VCI-MID (for AAL3/4) or VC (for AAL5) to the HNIC buffer. The host reads the Start-address and VCI-MID or VC, and then starts processing the CPCS-PDU frame. The second half of the HNIC buffer will be used by the host to store its information to the NI. The host processes the reassembled messages then returns the pointer (the cell's address inside the HI) of each cell body to the HNIC buffer to be reused again by another message. The REP will fetch these pointers and then store them at the tail-of-the-CB.

The host is responsible for accepting new connections after it negotiates with other hosts. These connections have their own VCI-MID or VC and the REP in the NI should be informed about these new connections. These information will be delivered to the NI through the second half of HNIC.

In case the host CPU is busy doing other processing while different cells keep arriving from the network, the REP could send a buffer status message to the host when that the HI buffer is getting full.

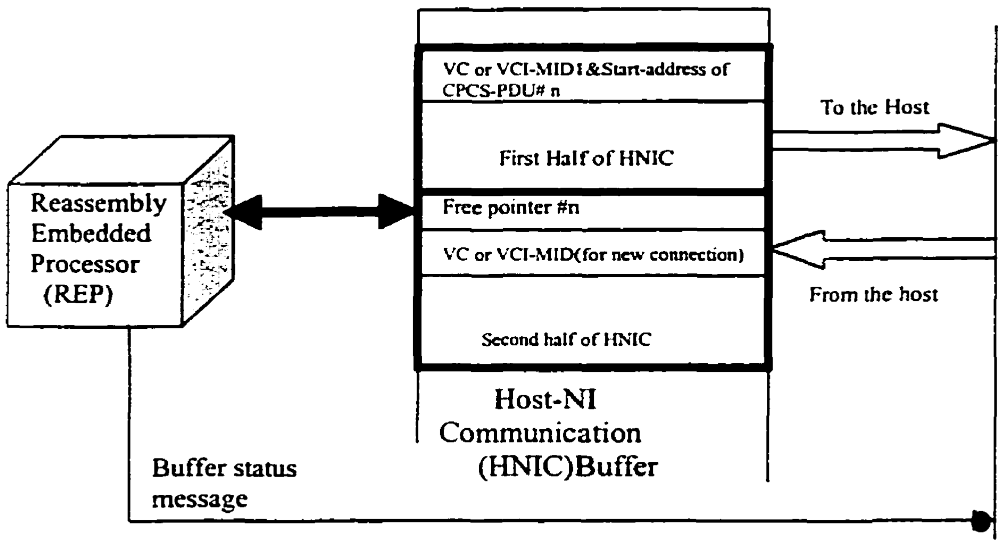

VC or VCI-MID1&Start-address of CPCS-PDU# n
First Half of HNIC
Free pointer #n
VC or VCI-MID(for new connection)
Second half of HNIC
Host-NI Communication (HNIC)Buffer
To the Host
From the host
Reassembly Embedded Processor (REP)
Buffer status message

Figure 3.3: Communication between host and NI Reassembly Embedded Processor

In the segmentation unit, the HNIC buffer will be used when the host decided to sends a block of data to the other ATM hosts. The host CPU sends the CPCS-PDU frame to the HI buffer follows with other information required to process the frame. This information includes VCI, VPI, and frame location inside the HI. The Segmentation Embedded Processor (SEB) must get the information from the HNIC in order to process a CPCS-PDU (Figure 3.4).

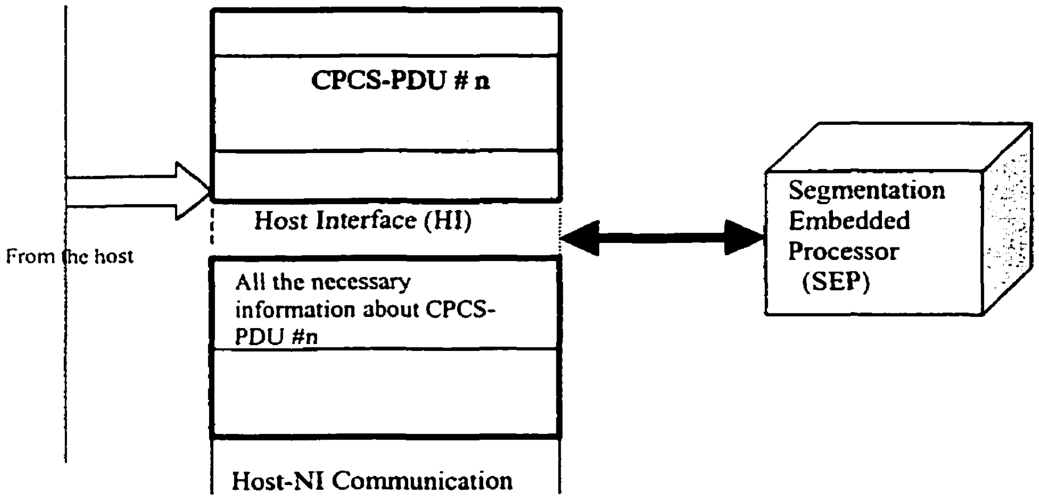

CPCS-PDU # n
Host Interface (HI)
From the host
All the necessary information about CPCS-PDU #n
Host-NI Communication
Segmentation Embedded Processor (SEP)

Figure 3.4: Communication between the host and the NI Segmentation Embedded Processor

3.2.1.2 Content Addressable Memory (CAM)

ATM network is connection oriented. The connection is set up between two hosts with specific identifiers (i.e. VCI and VPI). The cells that transmit over a connection should have the same identifiers. The receiver side can recognize all the arrival cells that have the same identifier are related to the same message and it should be reassembled together. In this case, the receiver must multiplex the arrival cells to its related message according to the cell's identifier. The REP should know about all the identifiers that the host has made in order to match it with the identifier of the arrival cells. The Reassembly unit is then enabled by Content Addressable Memory (CAM) to store all these identifiers; VCI-MID for AAL3/4 [DBru93, STraw93 and Ahme94] and VCI and VPI [Desi95, CKim98 and RHosb99]. With each identifier, the CAM also stores the location (Start-address and End-address) of each message inside the HI buffer to help the REP link together the arrival cells which have the same identifier by using a linked list mechanism to reconstruct the messages. We will describe the linked list mechanism later.

We chose a part of the NI's memory to represent CAM where finding a match with the CAM-base memory was achieved by fetching each location and then comparing it with the arrived cell's identifier until a match was made.

3.3 Processing the Reassembly Function.

The main function for the receiver section of the NI is to reassemble the arriving cells into a complete message by linking together the cells that have the same identifier in the HI buffer. Each cell has its own header(s) to carry the cell's information. The header includes information such as Payload Type (PT), which helps the receiver side to recognize the cell type if it is the Beginning Of Message (BOM), or Continuation Of Message (COM), or End Of Message (EOM) of the CPCS-PDU, or maybe a Single Segment Message (SSM) of the CPCS-PDU.

Other useful information carried by the ATM cell header is the cell identifiers. The REP reads the arrival cell's identifiers to be able to distinguish to which CPCS-PDU frame that the arrival cell should reassembled with. For AAL3/4, the 26 bit identifiers VCI and MID, which are located in ATM and AAL headers respectively, will be checked with the CAM entries in order to multiplex the arrived cell to its related CPCS-PDU. The AAL 5 has 24-bit identifier VC (VPI and VCI), located in the ATM header, that needs to be matched with the CAM entries that contain the active VC.

3.3.1 AAL3/4

As soon as the ATM cell arrives at the NI, the ATM header, AAL header, and AAL trailer will be processed. The 26-bit VCI and MID are masked from the ATM header and AAL header, respectively, in order to match these identifiers with the CAM entries. After the match is found, the REP reads the head of the CB in order to get the address for a free location inside the HI for the arrived cell's body. The two bits of the PT field inside the AAL header check to determine if the type of the arrived cell is BOM, COM, EOM, or SSM of a CPCS-PDU. After getting a match and the type of the cell, the REP then needs to move the cell body from the LI to the HI buffer.

3.3.1.1 Data Movement

In the Programmed I/O method, the REP handles all the procedures for moving the cell payload from LI buffer to HI buffer. Every single clock cycle, the processor loads 32-bit to its register and then during the next cycle, store these 32-bit to a specific place inside the HI buffer until all 44 bytes have been moved to the HI buffer (Figure 3.5).

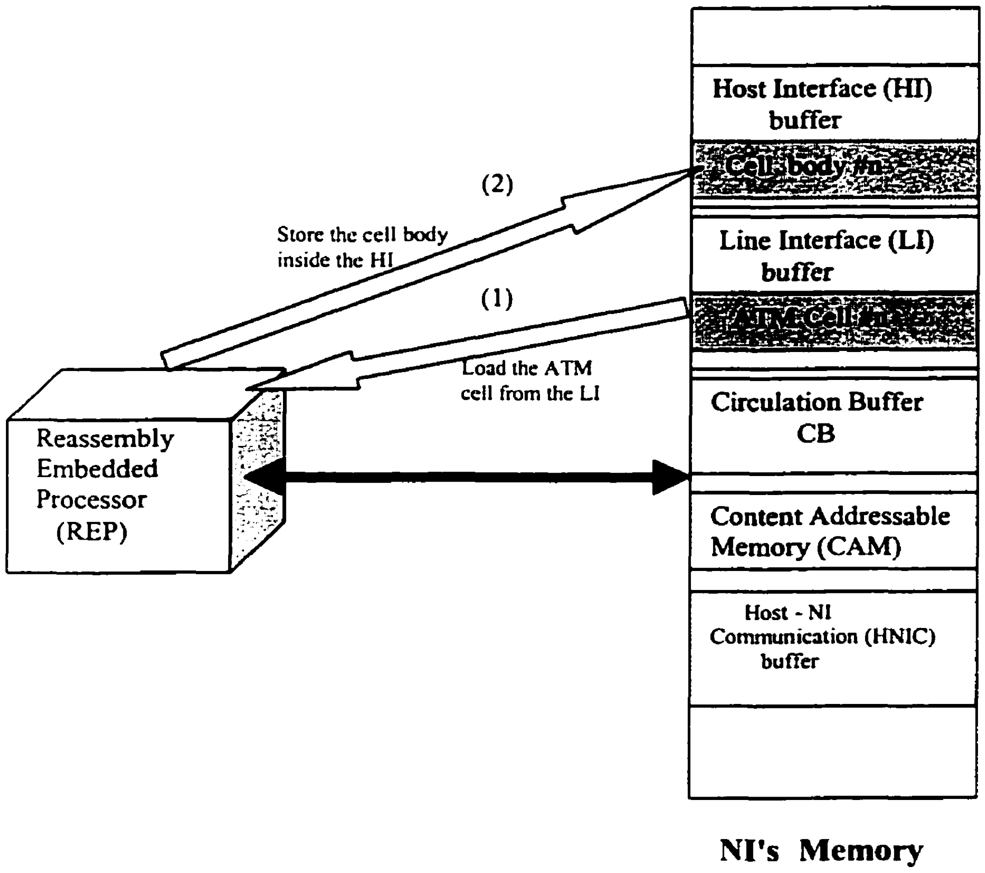

Host Interface (HI) buffer
Cell body #n
Line Interface (LI) buffer
ATM Cell #n
Circulation Buffer CB
Content Addressable Memory (CAM)
Host - NI Communication (HNIC) buffer
NI's Memory
(2)
Store the cell body inside the HI
(1)
Load the ATM cell from the LI
Reassembly Embedded Processor (REP)

Figure 3.5: Programmed I/O approaches for data movement.

In the other method, i.e. we use the DMA approach for data movement, the DMA is initiated by embedded processor to move the cell payload while the embedded processor remains in halt mode. The initiation of DMA needs two instructions. The first instruction includes the address from where the data has to be read, and the second instruction to inform the DMA about the destination location where the cell payload should be store (Figure 3.6). The amount of data that has to be transfer is not required since it is known in NI application.

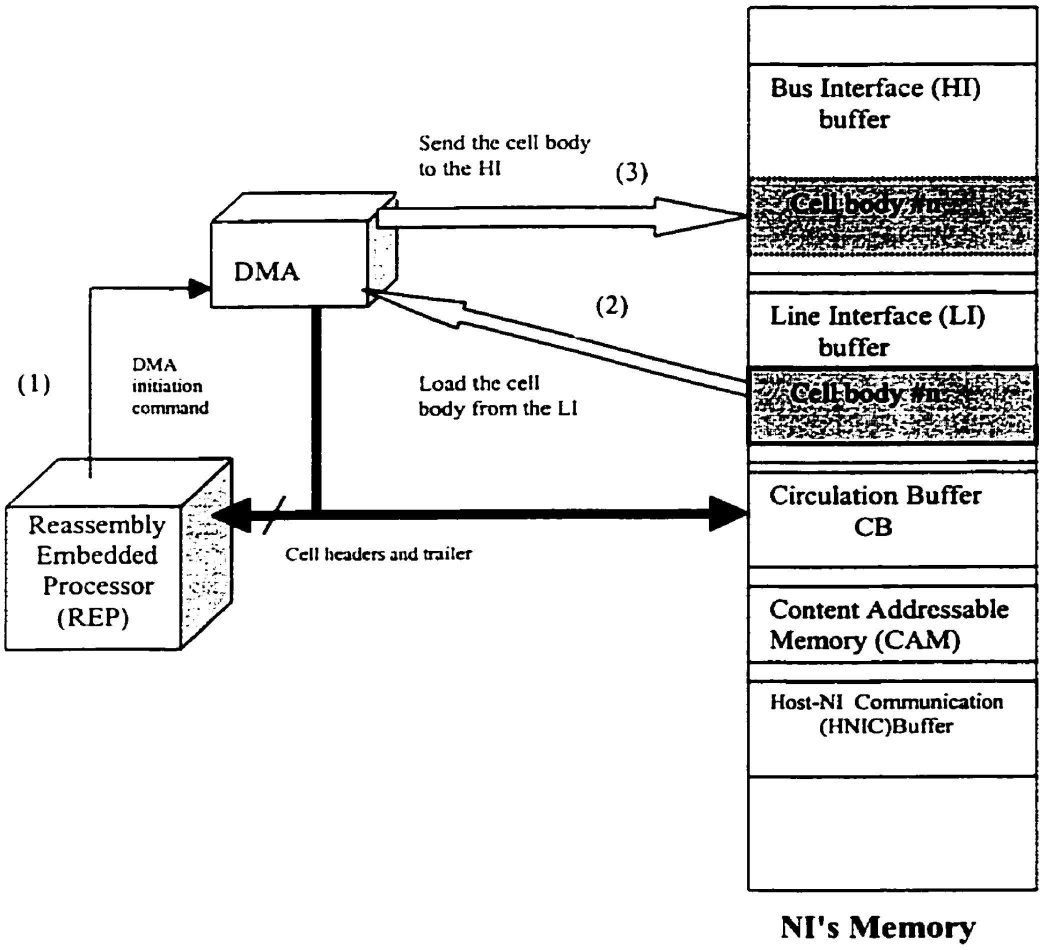

Bus Interface (HI) buffer
Send the cell body to the HI
(3)
Cell body #n
DMA
(2)
Line Interface (LI) buffer
(1)
DMA initiation command
Load the cell body from the LI
Cell body #n
Circulation Buffer CB
Reassembly Embedded Processor (REP)
Cell headers and trailer
Content Addressable Memory (CAM)
Host-NI Communication (HNIC)Buffer
NI's Memory

Figure 3.6: DMA approach for data movement with AAL3/4

After moving the cell body from the LI to the HI buffer, the REP then needs to link the arrived cell to its related CPCS-PDU frame which is considered as a part of that frame. Our approach for reconstructing the message is using a link-list mechanism.

3.3.1.2 Linked List Mechanism.

The multiplexing of the cells can be done according to VCI-MID, where each arrival cell which has the same VCI-MID should be linked together to get a complete message. Each VCI-MID in the CAM entries provided with two pointers, a head of the linked list pointer which holds the address of the first arrived cell, and the tail pointer which holds the address of the last cell that arrived for the specific VCI-MID. The linked list is useful for reconstructing the message from the ATM cells to a list of nodes, where each node has a cell body and pointer to next node (Figure 3.7).

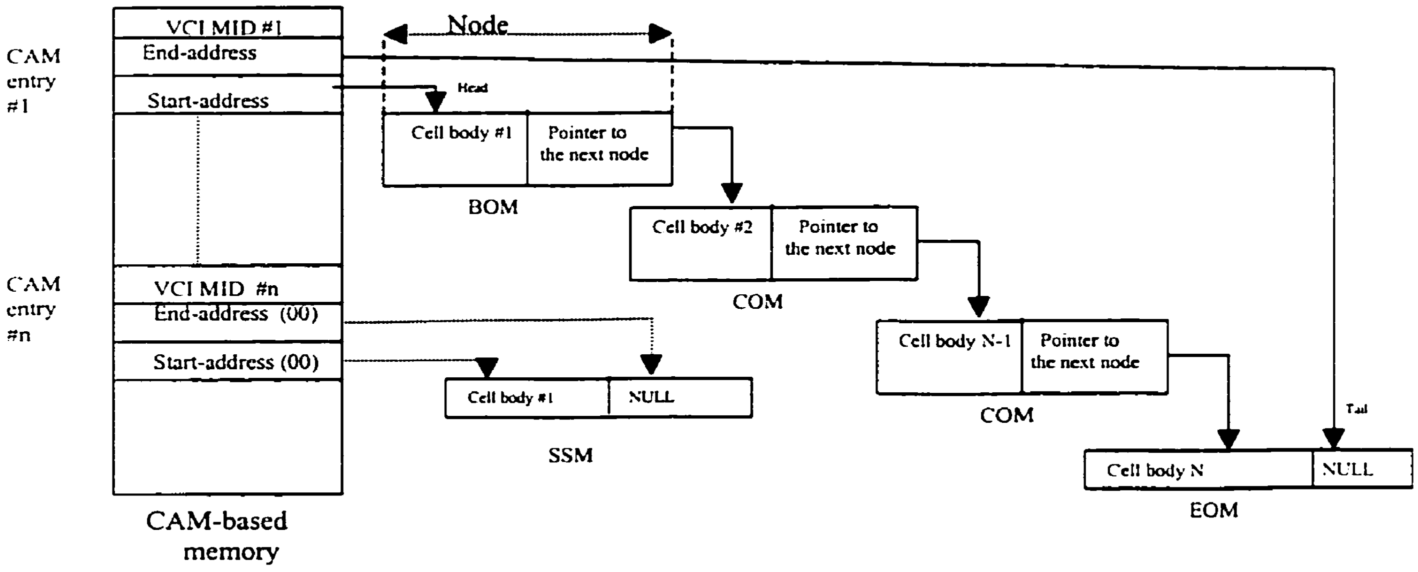


Figure 3.7: Linked list data structure.

After inserting a new entry in the CAM, all pointers (Start-address and End-address) are zeros. Changing these pointers depends on the processing that the arrival cell needs because each arrival cell may require different processing in the linked list approach than others, depending on its Segment Type (ST). In some cases, the ST indicates that the arrived cell is BOM (ST='10'). The REP then needs to create a new linked list for this CPCS-PDU frame by inserting the Start address and End-address in the CAM with this VCI-MID. The Start-address refers to a head of the linked list (the address which is loaded from CB for this cell). The End-address refers to the tail of the linked list which is the Null's address (Node's pointer), located at the end of the cell body. Thus, the linked list with one node was created for the arrived VCI-MID. The procedure of adding the new nodes (cell body and its pointer) at the end of the existing linked list after the match between VCI-MID of the arrival cell and the one in the CAM entries is as follows: Make the old node pointer point to the current node, and the current node pointer point to NULL, then store the NULL's address of the current node at the CAM referring to the new end of the list.

We have discussed the idea of adding a COM (where the ST = '00') in the existing linked list, now let us see how the same approach can be applied in the HI buffer. The End-address in CAM, which is referring to Null value of the previous cell, is read by REP and then stores the address of the current cell in the same place where the NULL value is of the previous cell. In this case the current cell was attached to the previous cell for the same VCI-MID, then store the NULL value at the end of the current node (Figure 3.8). Finally the NULL's address is stored in the CAM with the same VCI-MID which refers to the End-address of this VCI-MID.

When the ST refers to EOM (ST ='01'), we also need to add this node at the end of the list following procedure which we used for COM. With EOM there is no need to extend the linked list further because it is the last cell of the VCI-MID, and there is no need to store the End-address in the CAM. When the PT refers to SSM (ST='11') which indicates that there is only one cell for the VCI-MID, move the cell body from LI to the HI buffer . Finally store the NULL value at the end of the cell body where there is no need to update the End-address or Start-address at CAM because no more cells will be arriving after this cell.

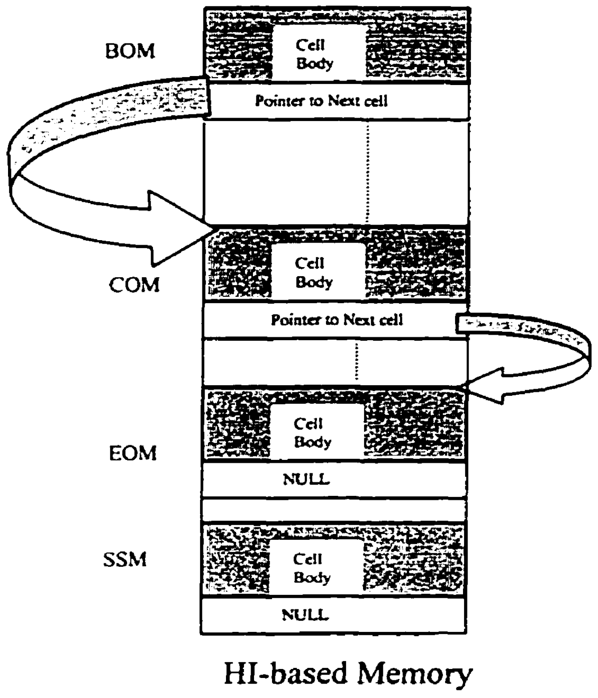


Figure 3.8: Linked list structure

3.3.2 Reassembly AAL5

The Reassembly function for AAL5 needs only to process the ATM header where the AAL header and AAL trailer do not exist for an ATM cell of type AAL5 [ITUT93]. Extract 24-bit VC (VPI and VCI) from ATM header and match it with CAM entries which contain the active VC [Desi95, CKim98 and RHosb99]. The ATM cell header holds the PT field which is useful to find whether the type of arrival cell is first, middle, last, or a single cell in a CPCS-PDU. When the PT is '0,' this indicate that the arrival cell is BOM or COM, otherwise EOM or SSM. The REP uses the Start-address located in the CAM to differentiate between the BOM and COM. If the PT is '0' and the Start address is '0' too, then this message is BOM. Otherwise, it is COM. The same approach will be applied for SSM and EOM.

The linked list data structure for AAL5 Reassembly function for arrived cells is processed in the same manner as for AAL3/4. For the data movement, the main difference between AAL3/4 and AAL5 is the size of the cell body. AAL5 has 48 bytes in its cell body which needs 12 cycles (48 Byte / 32-bit bus width) to move a complete cell body from one location to another. As the AAL5 has no AAL header and trailer, the processing requirement for AAL5 is less than to AAL3/4.

3.4 ATM Segmentation Function Processing

As the host transmit a CPCS-PDU frame to the other end, it moves the frame to the HI buffer of the Segmentation unit. Also, other necessary information is required to be sent by the host, such as the VCI and the location of the CPCS-PDU frame inside the HI buffer of the Segmentation unit, i.e. the start and end address. This information should be sent to HNIC for each CPCS-PDU frame.

3.4.1 Segmentation AAL3/4

The SEP reads the information which is available in the HNIC in order to start generating the first four bytes of ATM header (for the particular CPCS-PDU) and to calculate the size of the CPCS-PDU in order to determine the PT and sequence number (SN) fields in the AAL header. After the ATM header and AAL header have been generated, they are written into the SEP's register to be sent with each segment part of the CPCS-PDU frame (each segment is 44 byte). The SEP needs to change the PT and the SN for each leaving cell (i.e. the PT of the BOM is '01', COM is '00' and SN should be incremented for each leaving cell that have same VCI-MID).

For data movements using the programmed I/O, the embedded processor moves 44 bytes (as ATM cell body for AAL3/4) from the HI to the LI. In addition, the headers will be transferred from the SEP's register to LI. After moving the headers and the cell body, the SEP then generates the last byte of the AAL trailer which contains the actual length of the cell body, and then sends the trailer from the SEP's register to the LI (Figure 3.9).

For data movement using the DMA, the Embedded processor initiates the DMA to move the 44 bytes from the HI buffer to the LI buffer. The ATM cell header and trailer are moved from the SEP's register to the LI buffer. The initiation of DMA needs two instructions as discussed previously.

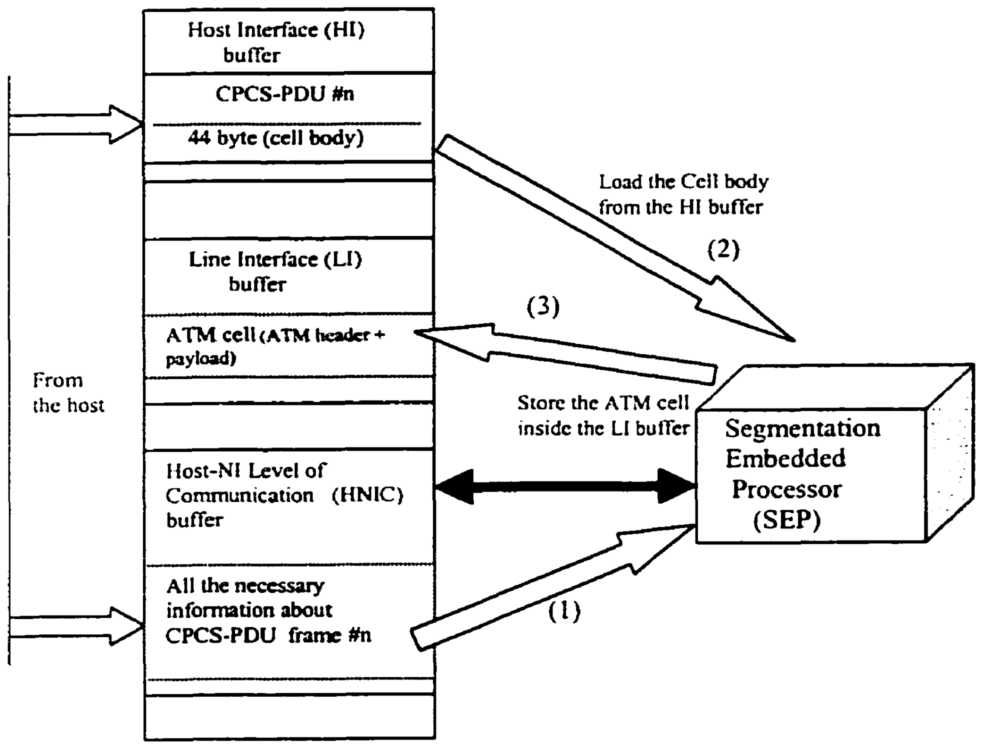

Host Interface (HI) buffer
CPCS-PDU #n
44 byte (cell body)
Line Interface (LI) buffer
ATM cell (ATM header + payload)
From the host
Host-NI Level of Communication (HNIC) buffer
All the necessary information about CPCS-PDU frame #n
Load the Cell body from the HI buffer
(2)
(3)
Store the ATM cell inside the LI buffer
Segmentation Embedded Processor (SEP)
(1)

Figure 3.9: ATM Segmentation

3.4.2 Segmentation AAL 5

The SEP needs to generate the ATM header for the CPCS-PDU frame and that header will be sent with each outgoing cell. Also there is no need to calculate the sequence number and the size of body for each leaving cell. Each BOM, COM, EOM or SSM cell needs to change the PT field in the ATM header to either '0' or '1' ('0' for BOM and COM, '1' for EOM and SSM). The only different between AAL 5 and AAL3/4 for data movements is the payload size of AAL 5 which is 48 bytes. With less headers and trailer, AAL5 requires less processing cycles than AAL3/4.

3.5 Simulation Results

During the simulation, we have measured the amount of processing required for ATM network interface protocols and for data movement. Different ATM cells have been delivered to the simulator and the number of instructions required for the Reassembly functions processing is measured for AAL3/4 and AAL5 (Table 3.1 for AAL3/4 and Table 3.2 for AAL5). After the embedded processor finishes processing one ATM cell, it then fetches the new connection identifier or a pointer that was sent by the host through the HNIC. Also, the NI needs to send the VC or VCI-MID with its Start-address to the HNIC after reassembling the CPCS-PDU frame in the HI. The amount of the execution that the processor takes for different types of operations for ATM Reassembly has been analyzed during this simulation (Table 3.3). The percentage measurement was taken when the processor execute the EOM, i.e. the result show the upper band of the execution rate since the EOM is required more processing than other type of messages.

Instruction	**First cell**	**Last cell**	**Description**
Load	3	3	Load the ATM header, AAL header, and AAL trailer from LI buffer.
Load	1	1	Load a space for the incoming message inside the HI buffer by reading the head of the CB.
Load	1	1	Load a VCI-MID from CAM to match it with incoming VCI - MID .
Load		1	Load the start-address from CAM to be stored in HNIC Buffer.
Load	-	1	Load the End-address from the CAM to get the NULL's address
Load	1	1	Load the pointer or VCI-MID from HNIC buffer
Store	2	-	Update the CAM by two entries a head and tail of the linked list
Store	-	2	Store the VC and start and End-address to the HNIC
Store	-	1	Store the address of incoming message in the previous message to be a pointer to the incoming message.
store	1	1	Store Null value at the end of node.
Store	1	1	Store the new pointer in the tail of the list of the CB, or store VCI-MID in the CAM
Arithmetic	4	3	add, addi
Logic	3	3	and
Branch	4	5	Condition branch
Total	21	24	

Table 3.1: The number of the Reassembly instructions needed to process an ATM message for AAL3/4.

Instruction	First cell	Last cell	Description
Load	1	1	Load the ATM header from LI buffer
Load	1	1	Load a space for the incoming message inside the HI buffer by reading the head of the CB.
Load	1	1	Load the Start-address from the CAM to be stored in HNIC Buffer.
Load	-	1	Load the End-address from the CAM to get NULL's address
Load	1	1	Load a VCI-MID from CAM to match it with incoming VCI-MID
Load	1	1	Load the pointer or new VC from HNIC buffer.
Store	-	2	Store the VC and Start-address to the HNIC buffer.
Store	2	-	Update the head and tail of the list in side the CAM.
Store	-	1	Store the address of incoming message in the previous message to be a pointer to the incoming message.
Store	1	1	Store Null value at the end of current cell.
Store	1	1	Store the new pointer in the tail of the list of the CB. Or Store the loaded VC in CAM
Arithmetic	3	2	add, addi
Logic	2	2	and
Branch	5	5	Condition branch
Total	19	20	

Table 3.2: The number of the Reassembly instructions needed to process an ATM message for AAL5.

Operation type	Processing Percentage rate AAL3/4	AAL5
Load	33.3	25
Store	20.8	30
Arithmetic and logic operation	25	20
Conditional branch	20.8	25
Reading/writing from/to HNIC	12.5 *	15 *
The liked list data structure	16.6 *	20 *
* Load, store and arithmetic instructions are involved		

Table 3.3: Contains the percentage of the processor power for ATM Reassembly.

The number of instructions involved in processing the Segmentation functions is shown in Table 3.4 and Table 3.5. Table 3.6 shows the percentage of each instruction used in the Segmentation function processing. Such result are measured for the processing of BOM which required more processing than any other message type, since its required to generate the cell headers and trailer for each BOM. That is, the upper bound of the processing rate is shown in Table 3.6. Also no data movement is involved in this calculation.

Instruction	First cell	Last cell	Comment
Store	3	3	Moving ATM header, AAL header and AAL trailer data from microprocessor 's register to network line.
Arithmetic	13	5	add, addi
Logic op	1	-	and
Branch	2	2	
Total instruction	19	11	

Table 3.4: The number of the Segmentation instructions needed to process an ATM for AAL3/4

Instruction	First cell	Last cell	Comment
Store	1	1	Moving ATM header from microprocessor 's register to network line
Instruction	6	2	add, and
Branch	2	1	Condition branch
Total instruction	9	4	

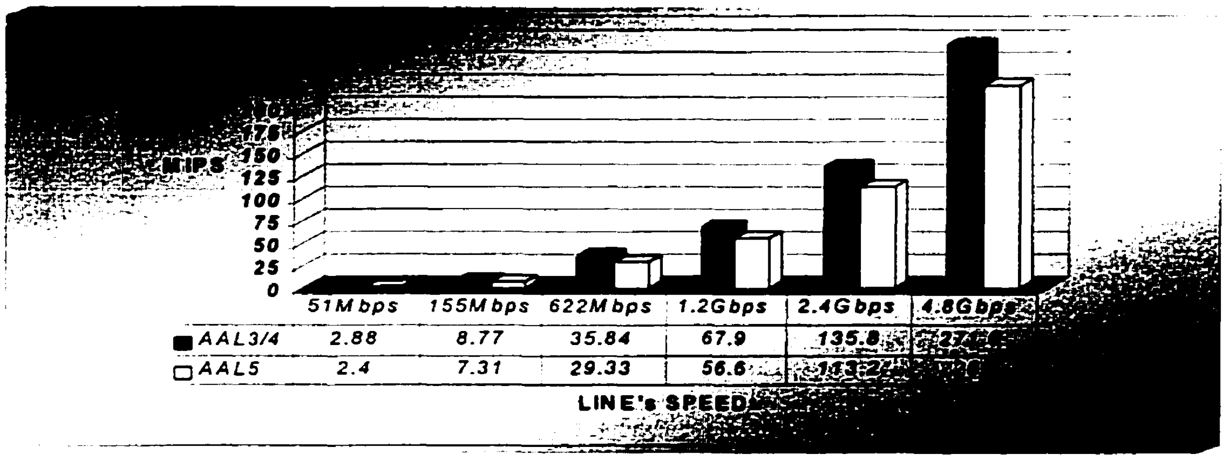
Table 3.5: The number of the Segmentation instructions needed to process an ATM message for AAL5.

Operation type	Processing percentage rate	
	AAL3/4	AAL5
Store	16	11.11
Generation ATM header	20 *	44 *
Generation AAL header	10 *	--
Generation AAL trailer	10	--
Arithmetic and logic operations	74	66.66
Conditional Branch	10.5	22
* Arithmetic and logic operations are involved		

Table 3.6: Contains the percentage of the processor power for ATM Segmentation

After calculating the amount of processing required by REP to process each ATM cell, we have measured the amount of processing that the REP should be performed in order to support different transmission line speed.

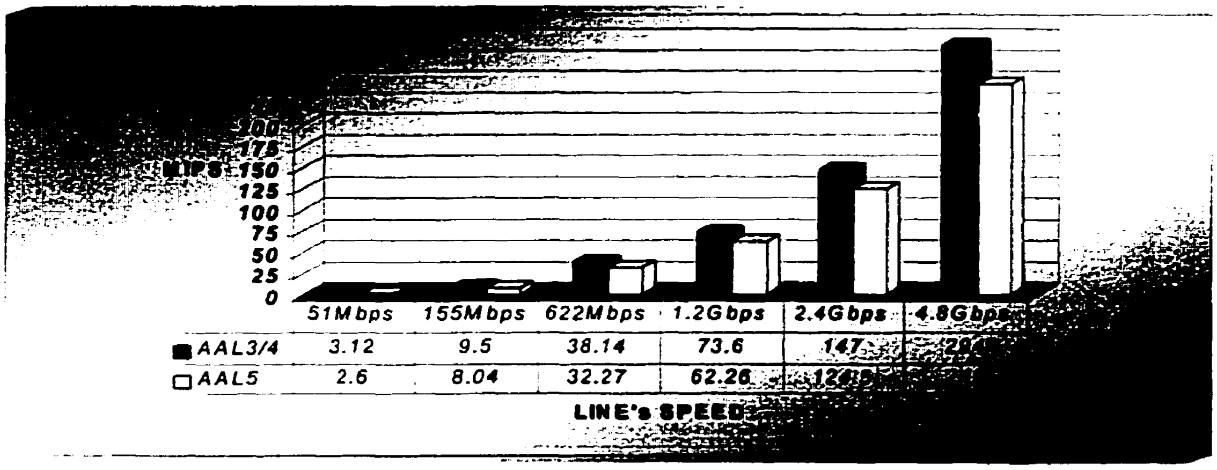
Figure 3.10 shows these results in million Instruction per Second (MIPS). Hence the REP is considered to be a RISC core, every instruction can be processed in one cycle. Therefore, the results shown in Figure 3.10, can be represent the required speed of the RISC core in terms of MHz. As we measured the upper bound processing for ATM cells, the results we present in Figure 3.10 represents the maximum REP clock rate to process different transmission lines.



	51Mbps	155Mbps	622Mbps	1.2Gbps	2.4Gbps	4.8Gbps
■ AAL3/4	2.88	8.77	35.84	67.9	135.8	[illegible]
□ AAL5	2.4	7.31	29.33	56.6	113.2	[illegible]

Figure 3.10: Reassembly function processing

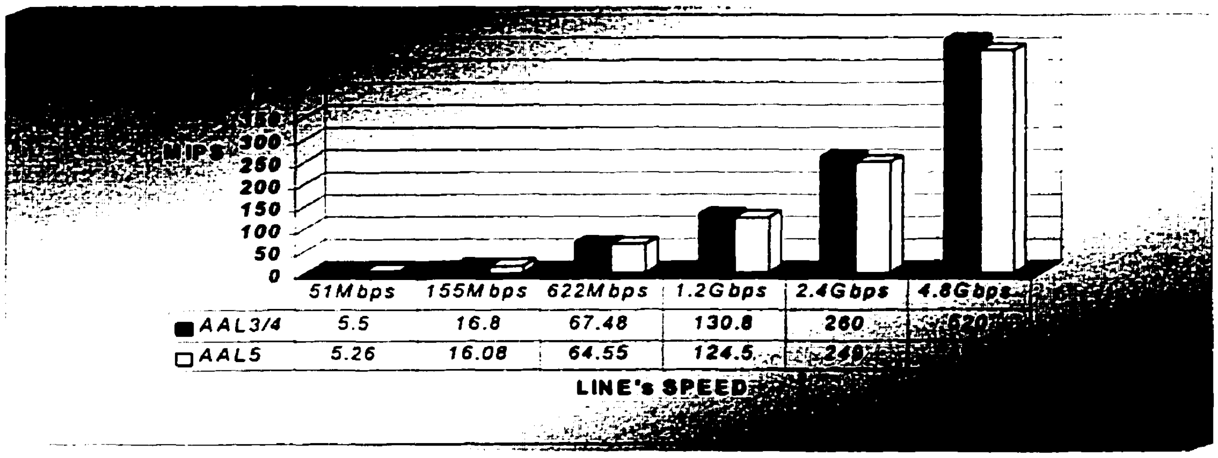
If the REP is involved in the DMA controller initialization, the results are very close to that without data movement and specifically when the transmission line speed is low (below 622Mb/s) Figure 3.11. However, as the speed of the transmission lines get higher, the amount of processing required for initialization of the DMA controller becomes significant.



	51Mbps	155Mbps	622Mbps	1.2Gbps	2.4Gbps	4.8Gbps
■ AAL3/4	3.12	9.5	38.14	73.6	147	29[illegible]
□ AAL5	2.6	8.04	32.27	62.26	12[illegible]	[illegible]

Figure 3.11: Reassembly with data movement using DMA

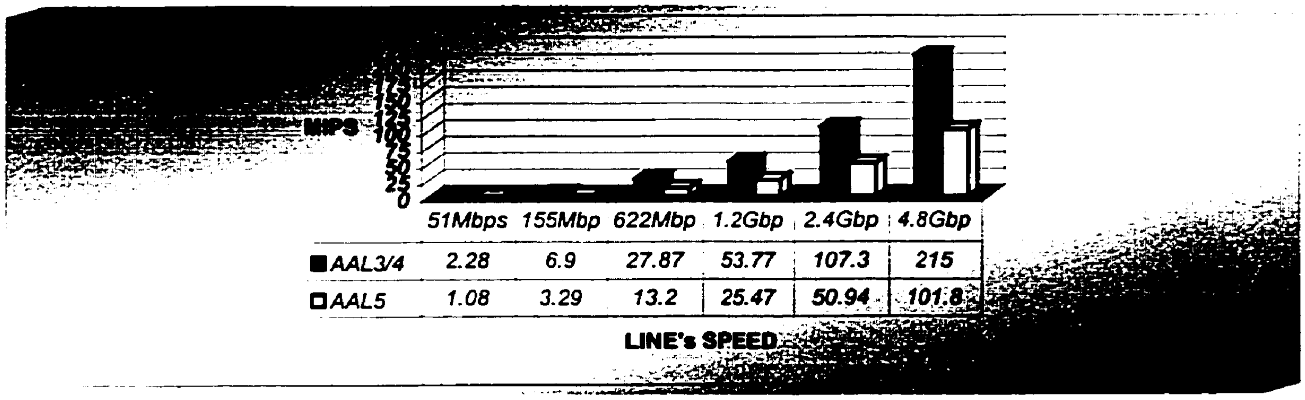
We have simulated the amount of protocol processing with the data movements using the programmable I/O technique. We found that the REP processing is become higher than that in the previous simulation where the DMA initialization is used (Figure 3.12).



	51Mbps	155Mbps	622Mbps	1.2Gbps	2.4Gbps	4.8Gbps
■ AAL3/4	5.5	16.8	67.48	130.8	260	520
□ AAL5	5.26	16.08	64.55	124.5	249	[illegible]

Figure 3.12: Reassembly with data movement using programmed I/O

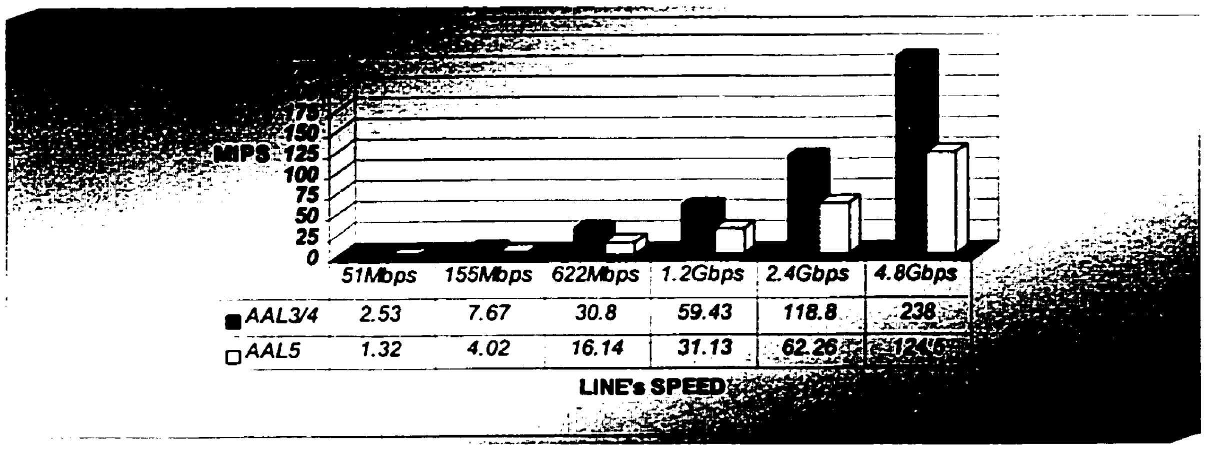
The amount of processing for Segmentation Function that the SEP should execute is calculated in the same manner as for Reassembly Functions (Figure 3.13). Clearly the RISC core clock rate is less than that for Reassembly function since the amount of processing for Segmentation function is less than that for Reassembly.



	51Mbps	155Mbp	622Mbp	1.2Gbp	2.4Gbp	4.8Gbp
■AAL3/4	2.28	6.9	27.87	53.77	107.3	215
□AAL5	1.08	3.29	13.2	25.47	50.94	101.8

Figure 3.13: Segmentation function processing

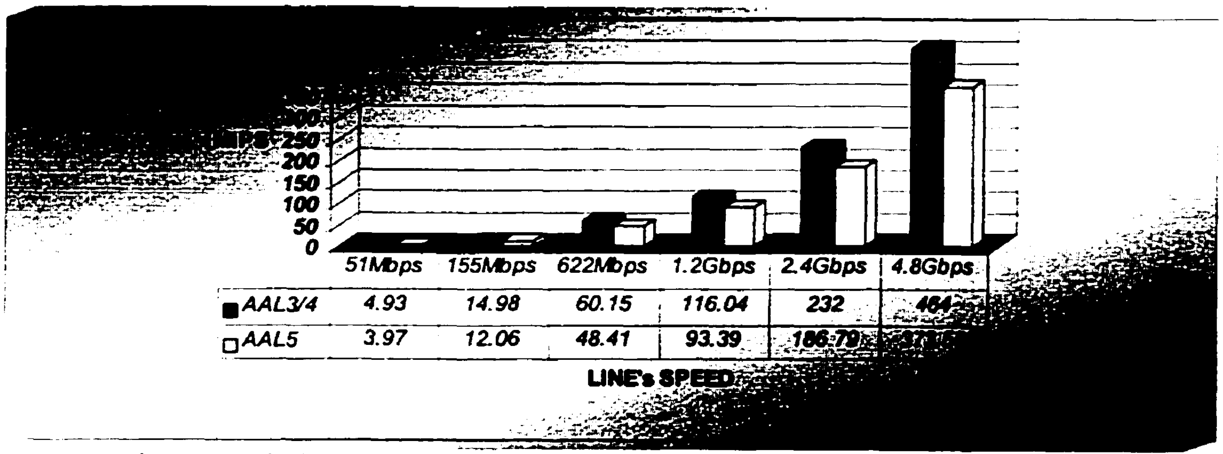
Figure 3.14 shows the amount of processing that is needed for ATM Segmentation where the SEP process the DMA initialization.



	51Mbps	155Mbps	622Mbps	1.2Gbps	2.4Gbps	4.8Gbps
■AAL3/4	2.53	7.67	30.8	59.43	118.8	238
□AAL5	1.32	4.02	16.14	31.13	62.26	124.5

Figure 3.14: SEP processing amount for Segmentation function and with DMA initialization

As we process the data movement using programmable I/O technique, in addition to the processing of segmentation functions, the number of instruction processed by SEP is increased significantly (Figure 3.15).



	51Mbps	155Mbps	622Mbps	1.2Gbps	2.4Gbps	4.8Gbps
■ AAL3/4	4.93	14.98	60.15	116.04	232	464
□ AAL5	3.97	12.06	48.41	93.39	186.79	[illegible]

Figure 3.15: Segmentation with data movement using programmed I/O

3.6 Conclusion

The Segmentation Function requires less processing than Reassembly function because of the nature of the function, which is simpler than Reassembly. Generally, the Embedded processor core running on a lower clock rate will be more useful for the network interface where the cost of such core will be low. Hence, the Embedded processor core should be supported with a DMA controller. The processor that use a programmed I/O approach for data movements will process about 33 % more than the one using the DMA technique for data movements.

It is clear from the simulation result that a 1.2 Gbps ATM network interface can be achieved by using an embedded processor running at 74 MHz for Reassembly function processing, and 60 MHz for Segmentation function processing. These result are applied for supporting AAL3/4. The processing requirements for AAL5 are much less than with AAL3/4, about 63MHz and 32MHz for Reassembly and Segmentations respectively.

Chapter 4

VHDL Simulation for ATM NI

4.1 Introduction

The amount of processing required for the ATM network interface supporting different transmission line speeds was investigated. We found that the use of an embedded RISC core, run at 74 MHz, in ATM NI design could support up to 1.2Gbps transmission line. We know that a SPIM-based simulator gives an estimate of that processing, since the simulator does not simulate the real hardware that is usually used with NI design. This gives us an incentive to investigate a detailed simulator that simulates a real ATM NI. Such simulator uses the RISC core and other components that are required in the interface design such as the DMA, Content Addressable Memory (CAM), FIFO, CB, the transmission line interface, and the host interface buffer. With such a simulator, we can find the accurate results for RISC clock rate, RISC processing, and NI structure. We decided to use VHDL in our simulator because it is suitable and powerful to capture complex digital system design for both simulation and synthesis [KSKA96, DPERR98]. VHDL also has many features appropriate for describing the behavior of components ranging from simple logic gates to complete microprocessors and custom chips. The IEEE *1164-1993 standard* running over the Xilinx foundation version 1.5 was used in our simulation [Xilin98, Xilin99]

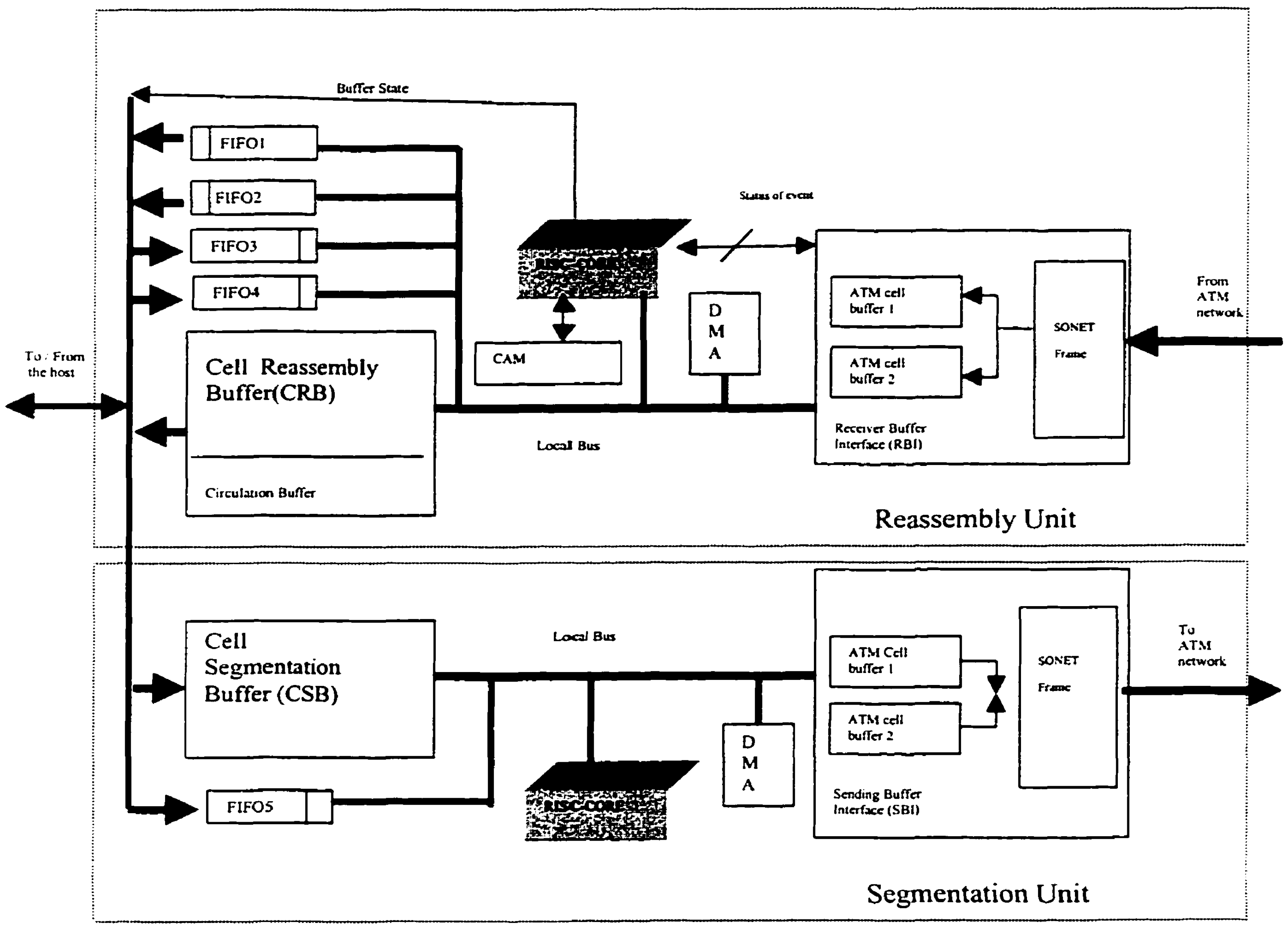
The VHDL-based model for Segmentation and Reassembly function of both AAL3/4 and AAL5, is developed and described in this chapter. The NI components and their operation will also been described in details. The chapter concludes with the VHDL simulator results.

4.2 ATM Network Interface Model

The NI model we proposed is partitioned into three parts: the communication line interface, the processing core, and the host bus interface (Figure 4.1). The processing core performs the NI functions such as Segmentation and Reassembly, the PDUs, VCI and VPI for AAL5, VCI and MID for AAL3/4, linked list scheme, cells copying and buffering.

The model has the architecture that can support high-speed lines for both AAL 3/4 and AAL 5 and it provides several features:

- Data movement using DMA.
- Two RISC-cores, one per direction (one for Segmentation unit and the other for reassembly unit), perform all functions related to the AAL3/4 and AAL5.
  - Using Content Addressable Memory (CAM) for virtual channel traffic [Gore97, STraw93], the CAM contains the active VC or VCI-MID connections to help the RISC in the Reassembly unit to reconstruct incoming cells to their PDU using the link-list scheme.



FIFO1: To carry VC or VCI-MID and Start address for each received a signaling cell.
FIFO2: To carry VC or VCI-MID and Start address for each received PDU.
FIFO3: To carry a new VC or VCI-MID to the receiver RISC.
FIFO4: To carry the free pointer space to the receiver RISC
FIFO5: To carry the necessary information such as VCI to the Transmitter RISC
DMA : Direct Memory Access
CAM : Contains Addressable Memory

Figure 4.1: ATM Network Interface Architecture

- To provide high flexibility in terms of exchange information between these RISC-cores and the host through the First-In-First-Out (FIFO) buffer. These FIFOs performed the following tasks:
    - FIFO1 carries information to the host such as VC or VCI-MID and a Start-address for each signaling message received.
    - FIFO2 carries important information to the host such as VC or VCI-MID and a Start-address for each received PDU.
    - FIFO3 carries the new VC or VCI-MID from the host to the RISC core at the Reassembly unit.
    - FIFO4 carries the free pointer from the host to the RISC core at the Reassembly unit.
    - FIFO5 carries information to the RISC core at the Segmentation unit, that needed to generate an ATM header and/or AAL header.
- NI buffers
    - Receiver Buffer Interface (RBI) is used to buffer two arrival cells and to deliver them to their destination
    - Cell Reassembly Buffer (CRB) is storing ATM cells (The payload part only).
    - Sending Buffer Interface (SBI) is similar to the RBI and it used to buffer up to two ATM cells until they delivered to the network.
    - Cell Segmented Buffer (CSB) is used to hold the PDU that it is received from the host. Such PDU will be segmented by the SEP and delivered to the SBI.

4.2.1 Data Movement

In the Programmed I/O, the RISC core retains control of the bus while data is moved. Programmed I/O is slow because there is too much unnecessary overhead for small transfers. There are also drawbacks to this approach, such as the RISC core being tied up moving data to or from the network interface. This affects the performance of the RISC by keeping the RISC core unavailable for other activities.

The use of DMA in NI is more efficient for NI applications than the programmed I/O. Therfore, a DMA is used in our simulator for data movemnet function [ECoop91, DBru93, CKim98 and RHosb99]. The DMA moves data from one location to another using its data register. The data moved from a source to the DMA's register and then storing it in its appropriate location Figure 4.2.

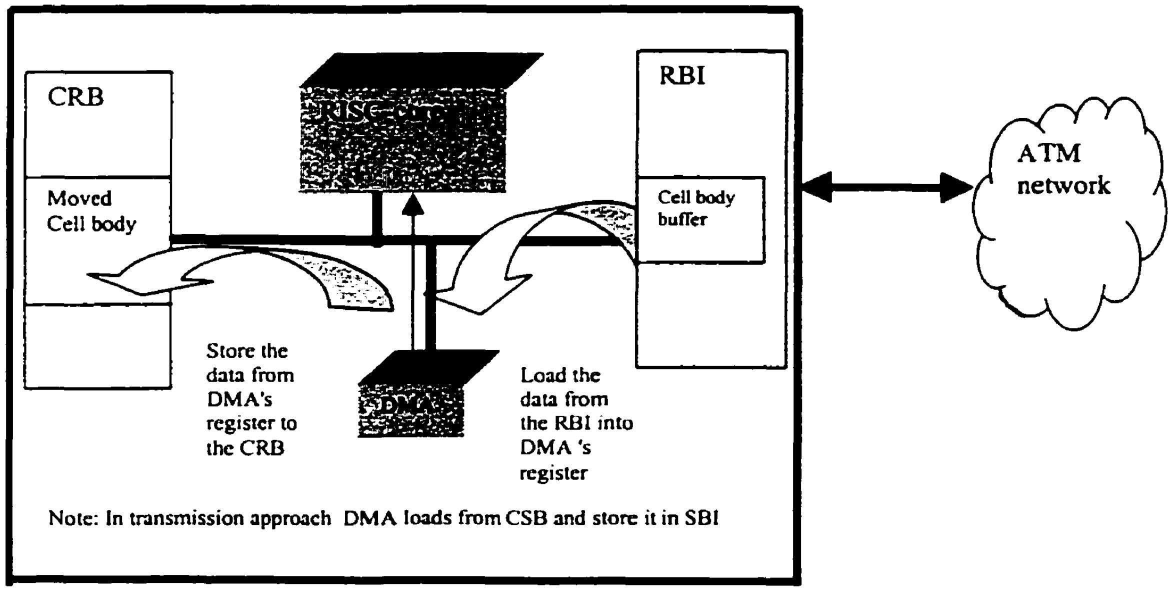

CRB
Moved Cell body
RISC-core
RBI
Cell body buffer
ATM network
Store the data from DMA's register to the CRB
DMA
Load the data from the RBI into DMA 's register
Note: In transmission approach DMA loads from CSB and store it in SBI

Figure 4.2: Block diagram of RISC-core with DMA

As the block of data is required to move between the RBI and the CRB, or between CSB and SBI, the RISC core will initiate and control the DMA. Since the local bus of the Reassembly and the Segmentation units is shared between the DMA and the RISC core, the RISC core will have to release the local bus to DMA to perform the data block transfer. Each transfer of a word consumes two cycles. In the first cycle, the DMA read the source buffer to get the word to the DMA's register. During the second cycle, the word will be moved from the DMA's register to the destination buffer. The DMA state machine will provide the read and write signals to source and destination buffers. Also it increments the address for the next location, where the next data is located, and store it in the appropriate location in the destination buffer (Figure 4.3). The schematic capture of the DMA stucture is showm in Appendix A Figure A.3.This process will continue until the whole cell will be completed. The VHDL based DMA has the state machine that required the following information:

(a) Block length (number of words to transfer).

(b) Direction (from CSB to SBI) / from RBI to CRB).

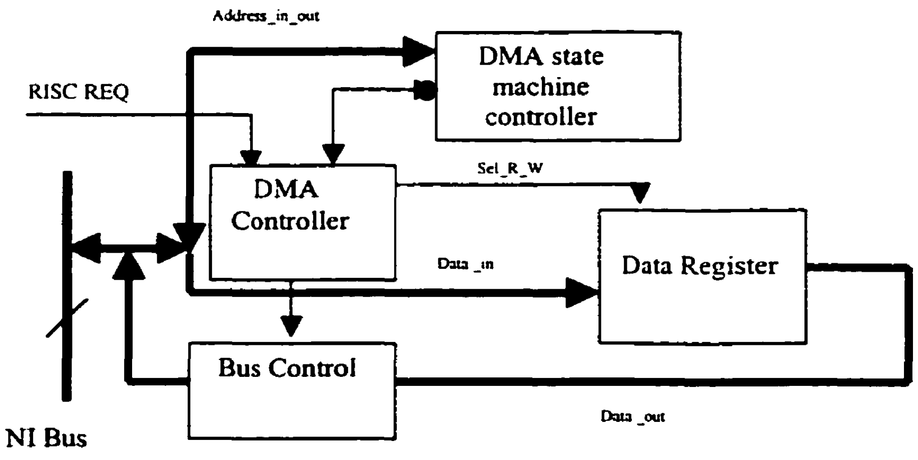


Figure 4.3: DMA structure

4.2.2 Content Addressable Memory (CAM)

CAM needs to contain the network addresses of all the active connections that the host has made with other hosts. Thus, the RISC can reconstruct the CPCS-PDU frame from the arrival cells using the addresses contained in the CAM. The VHDL based CAM was simulated as a Look-Up table for VC or VCI-MID (VC for AAL5 and VCI-MID for AAL3/4). With each CAM entry there are two pointers, a Start-address (head of the link-list) and End-address (the tail of the linked list). Figure 4.4 shows the CAM structure. The schematic capture of the CAM loction in the NI is shown in the Appendix A Figure A.4.

VHDL based CAM is implemented to have two kinds of processing. If there is any new entry needed to be stored in CAM, the first kind of processing is used to insert the new entry in the CAM. The write signal "1" is sent through the Sel_CAM signal bus by the RISC's controllor to replace data (adding new entry) in the CAM. The processor starts searching for the first location filled with Zeros (blank location) at any place in CAM and then replaces the first entry that has Zeros with the new address. The second kind of processing is used to find a match of the input data with the one in the CAM entries. If no entries of the CAM match the input data, a "miss" signal '0' is asserted to RISC. The RISC then considers the arrival cell as a lost cell (is not related to this host). If any entries of the CAM match the input data, the CAM produces a signal '1' indicating that the match was found. After finding the match, the processing continues reading the other signals to figure out the next procedure. There are two procedures after finding the match, either reading or writing the data from or to the CAM. In the first procedure, if the W_Data is '1', either the Start or end address is sent out according to the signal of

Sart_End_Add. If it is '1' the Start-address is sent out. Otherwise, the End-address is sent. In the second procedure, if the W_Data is '0,' the CAM writes either the Start or End-address according to the Sart_End_Add signal. If it is '1,' then The CAM writes the Start-address (where the match was found) otherwise writes the End-address.

Removing the CAM entries can be implemented simply by searching the match that is needed to remove from the CAM, and when it is found, replace it with Zeroes. We did not implement the removing entry from the CAM. Our intention is to calculate the amount of processing that is needed for each cell if the connection was active.

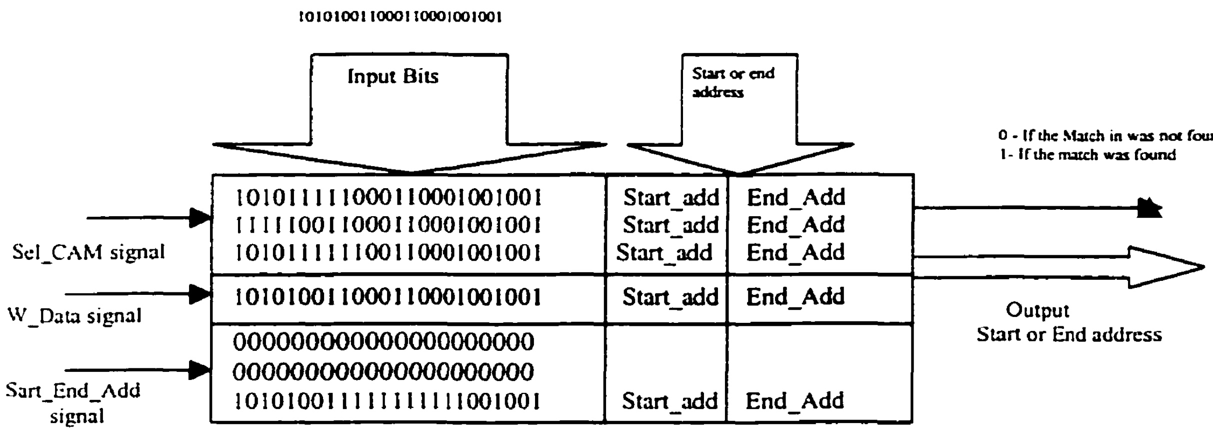


Figure 4.4: CAM Structure

4.2.2.1 Linked list CAM VHDL based

The linked list mechanism is similar to that used with SPIM simulator. The Start address and End-address were implemented to be in the same entry with the VC (if AAL5 is used) or the VCI-MID (if the AAL3/4 is used) (Figure 4.5). The Start and End Address are updated differently based on processing that is required to be performed by the NI and as the following:

1. Read the End-address from the CAM to find the end of the linked list (if we need to add a new node at the end).
2. Write the End address in CAM after receiving BOM and COM.
3. Write Start-address in CAM after receiving the first cell of CPCS-PDU. After receiving the last cell of the same CPCS-PDU, this start address will be read and sent to the FIFO2. That will tell the host a specific CPCS-PDU frame is reassembled. When the host reads that frame, the Start address will tell where the frame is in the NI local memory.
4. The PT for AAL5 can not tell if the arrived message is first or a continuation of message, because both have the same value "0." Therefore, the Start-address is used to distinguish whether the type of arrived cell is COM or BOM. If the Start-address equals "0" then the arrived cell is BOM. Otherwise, it is COM. The Start-address is also implemented in the case of SSM or EOM where both have value "1" in their PT. Therefore, if the Start-address is "0," the message is SSM. Otherwise, it is EOM.

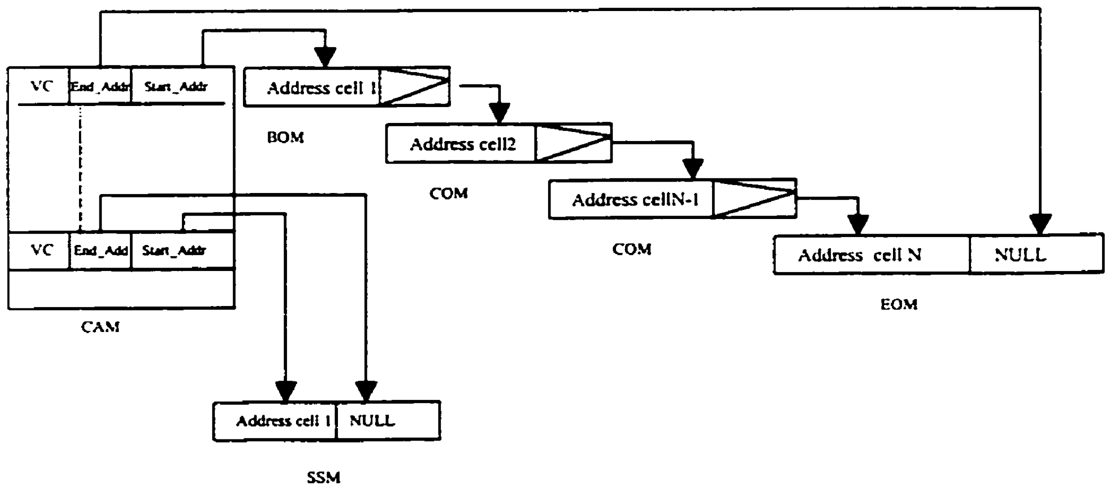


Figure 4.5: CAM architecture with its linked list mechanism

4.2.3 The NI's FIFOs

The NI communicates with the host through five FIFO buffers. These FIFOs were implemented as memory-based and the pointer of each FIFO is stored in the RISC's register. The RISC is able to reach any FIFO after reading its address.

However, the interrupt mechanism that happens during the exchange of information may effect the overall performance of NI or the host CPU. Interrupting the host CPU or RISC cores (SEP or REP) during their processing time will cost a certain amount of time in processing the interrupting task. The interruption of the host costs at least 15 microseconds on Sun SPARC station for line speed 155Mbps with the RISC speed is 50 MHz [ECoop91]. These interrupts reduce the performance of the processing power of the Host CPU.

In our simulator, we have eliminated the overhead processing of the host CPU caused by the above interrupt. This has been achieved by the following:

1. Instead of interrupting the host upon the arrival of each cell (i.e. 680 ns if the line speed is 622 Mbps), the RISC core will send the VC or VCI-MID and the Start-address to FIFO2. After accumulating the complete CPCS-PDU frame, the host then reads FIFO2. The host CPU starts fetching the CPCS-PDU frame from CRB for that specific VC or VCI-MID read from FIFO2 (Figure 4.6). The host then starts reading the first cell body until it reaches its pointer, which is placed at the end of the cell body. This pointer is important to locate the next cell in the CRB. The reading of the next cells will continue until the NULL value, which is placed at the last cell of the frame is located. By doing so, the use of the interrupt is eliminated and the associted time is saved.

The signalling messages will arrive at the NI and the NI should pass these messages to the host immediately. The Start address and VC or VCI-MID are sent to the FIFO1 and place the cell body in the CRB (Figure 4.6).

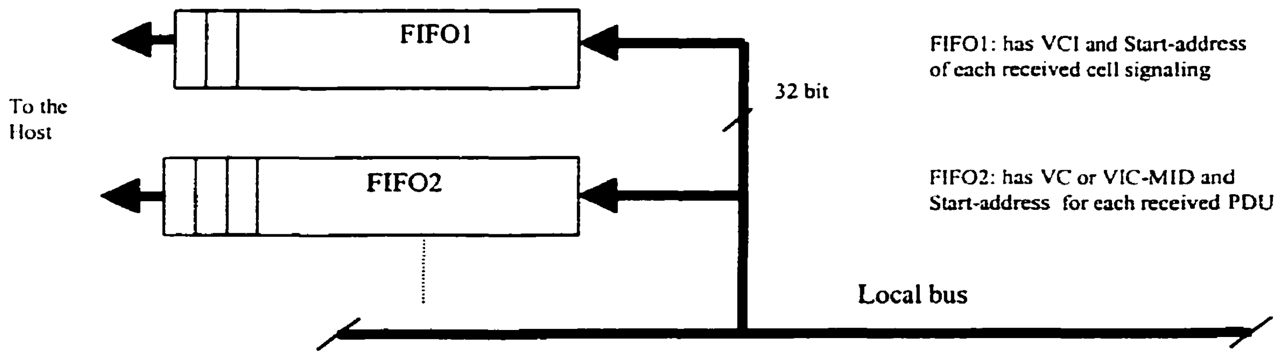

FIFO1
FIFO2
To the Host
32 bit
FIFO1: has VCI and Start-address of each received cell signaling
FIFO2: has VC or VIC-MID and Start-address for each received PDU
Local bus

Figure 4.6: The two FIFOs used to send the data from RISC processor to the host CPU.

2. The interrupt is used in our model in only one core when the host will be interrupted if the number of cells at the CRB occupy 90% or more of the CRB space. This interrupt will force the host processor to read some of the arrived messages in order to leave a space inside the CRB for other incoming cells.

There are no interrupts used when the host sends information to the NI. All information is delivered to NI through three FIFOs. Such delivering information to the NI can be described as following:

1. The host negotiates with other hosts whenever a new connection is required. After the connection is established between two hosts, there is a specific VC or VCI-MID will be allocated for specific connection. Either one should be used by the NI when a cell send to the other end. This VC or VCI-MID should be delivered on the cell header(s) where the NI processing them to multiplex the cells to its related VC or VCI-MID.

The host will use FIFO 3 to deliver VC or VCI-MID to the NI. As the cells arrive to the NI, the CAM entries will be updated by storing the VC or VCI-MID that was fetched from FIFO3 (Figure 4.7). The embedded RISC at the Reassembly unit will check the FIFO after finishing the Reassembly processing of each ATM cell (i.e. every 353 nonsescond, if the NI connected to line 1.2 Gbps)

2. After the host finishes reading the Reassembled message, the host sends the cell's address to the NI through the FIFO4. The NI will then read the free pointer address after finishing the processing required for the current ATM cell and then store the pointer back in the Circulation Buffer (CB) for later use.

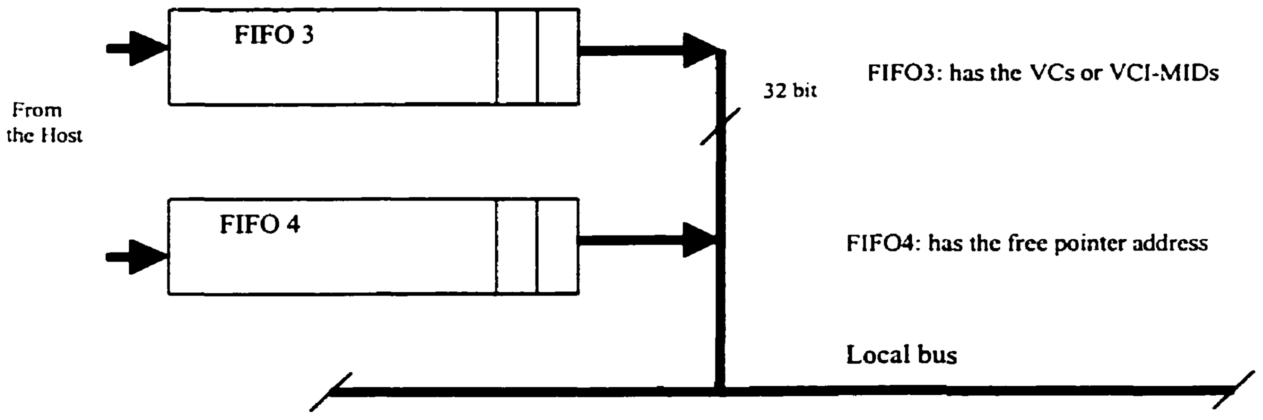

FIFO 3
From the Host
32 bit
FIFO3: has the VCs or VCI-MIDs
FIFO 4
FIFO4: has the free pointer address
Local bus

Figure 4.7: The two FIFOs have a VCs / VID-MID and free pointer address

3. When the host moves the PDU to the CSB, the host should notify the sender RISC by sending to FIFO 5 the location of the PDU frame, VC or VCI-MID, and other necessary information needed for segmented the CPCS-PDU frame (Figure 4.8).

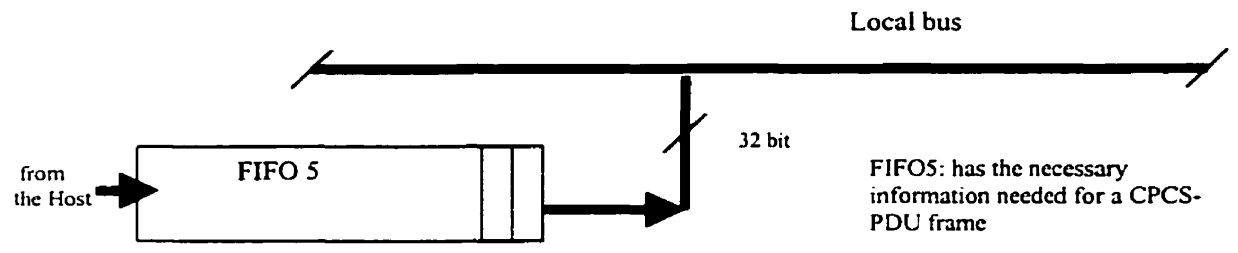

Local bus
32 bit
from the Host
FIFO 5
FIFO5: has the necessary information needed for a CPCS-PDU frame

Figure 4.8: The FIFO carries the information needed for Segment CPCS-PDU frames

4.2.4 The interface buffers

4.2.4.1 The Cell Receiver Buffer (CRB) and Cell Segmentation Buffer (CSB)

The NI has a CRB which is used to reassemble the cell bodies arriving from the network and store them until host is ready to process them. Each cell is represented in the buffer as 12 locations of memory, each with 32-bit word (44-byte and the pointer location) for AAL3/4, or 13 locations, each with 32-bit (48-byte and the pointer location) for AAL5. The size of the CRB buffer is 256 Kbytes. This buffer can hold of 4 CPCS-PDU payload where each payload may contain 64 Kbytes. Obviously, more payloads can be held if they are smaller than 64 Kbytes

The CSB stores the PDUs frames which are sent by the host to be segmented and sends them to the network line as ATM cells after adding its header(s) and/or trailer. This buffer can also hold 3 CPCS-PDU frames (each frame has 64 Kbytes for CPCS-PDU payload), in addition to the trailer and/or header for CPCS-PDU.

4.2.4.2 Receiver and Transmission Line buffers

When a cell arrives at the RBI, the FSM in the RBI will enable one of the two buffer locations to hold the serial bits arrived from the transmission line. A FSM will enable one buffer and can switch to another buffer after interrupting the RISC-core in the Reassembly unit. The RISC core will start processing the ATM cell header which is located at the top of its body (Figure 4.9). The RBI VHDL based is shown in Figure A.5 in appendix A.

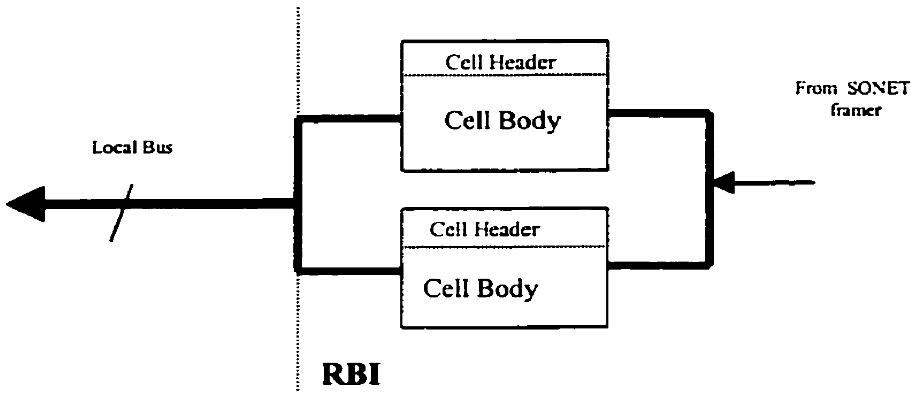


Figure 4.9: The buffer architecture in RBI

The SBI contains two buffers each of which hold one ATM cell. The sequential machine controls the SBI and allows only one buffer to be active and to receive data at a given time. The buffer will remain enabled until the complete ATM cell has been stored (Figure 4.10). The SBI VHDL based is shown in Figure A.6 in appendix A

The sequential machine will then allow the stored data to be sent out while it fills the other buffer.

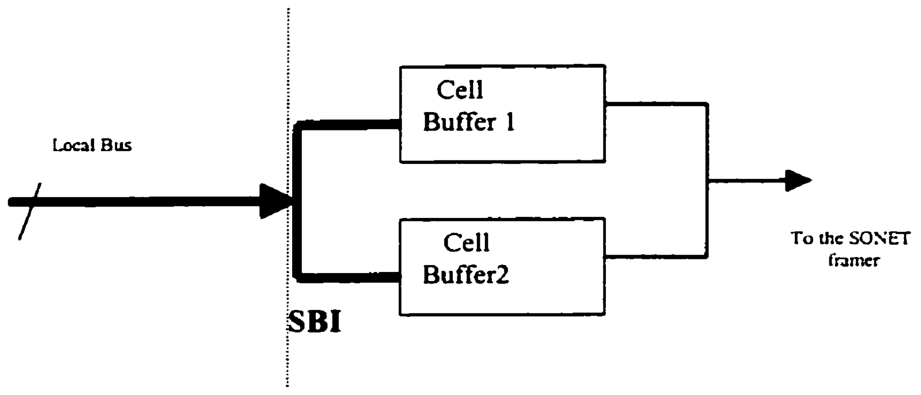


Figure 4.10: The SBI architecture

4.3 VHDL Simulation Results

Three stages pipelined RISC core, CAM, DMA, RBI, SBI, FIFOs, CRB, SCB, and CB have been used to simulate the ATM line interface. After testing the VHDL model of each component, a complete NI has been designed based on the model that presented before. All NI components are connected together with all the necessary connections, busses, and control lines (Figure A.1 in the appendix A). A testing process has been performed to check the functionality of the NI and to perform the performance evaluation that required for this research. We believe that by processing ATM cells with such RISC core based NI, we can measure the amount of RISC processing for different transmission line speeds.

4.3.1 Reassembly Function

By delivering different ATM cells to the simulated model and by investigating the waveform generated from the simulator, we are able to find the number of instructions needed to process a complete ATM cell for both AAL3/4 and AAL5. The results presented in Table 4.1 is for the NI when the DMA has the same RISC 's clock speed. The number of instructions is dependent on the type of the cell, i.e. BOM, EOM, COM, and SSM. The DMA needs 22 (11 load and 11 store) RISC instructions in order to move one payload body for AAL3/4 and 24 (12 load and 12 store) RISC cycle for AAL5. However, some of the RISC processing instructions needed to use the local bus, in order to send the NULL value at the end of the cell body during the processing of the link list mechanism. In this situation the RISC has to wait for several cycles until the DMA completes its job. The RISC wait cycles will reduce the NI performance by extending the execution time for each ATM cells by number of the RISC's wait cycles (Table 4.1).

Type of cells	**No. of instructions**		**No. of idle cycles**	
	AAL5	**AAL3/4**	**AAL5**	**AAL3/4**
Single Segment Message (SSM)	38	40	12	10
Beginning Of Message (BOM)	38	41	11	9
Continuation Of Message (COM)	38	41	12	9
End Of Message (EOM)	40	41	12	11

Table 4.1: number of instructions processed for Reassembly AAL5 and AAL3/4 messages (the DMA's clock has the same speed as RISC's clock).

To eliminate the RISC's idle cycles, we forced the DMA to finish its processing cycle in a shorter period of time than the first approach where the DMA has the same RISC's clock cycle. Therefore, the DMA clock rate is increased to run faster than RISC to allow the local bus to be available fro both DMA and RISC core. The DMA runs double the RISC's clock to complete moving 44 byte within 11 cycles and 12 cycles for 48 byte (Table 4.2). In this case, we eliminate all the idle cycles which could take almost 25% of the RISC's power.

Type of cells	**No. of instructions**		**No. of idle cycles**	
	AAL5	**AAL3/4**	**AAL5**	**AAL3/4**
Single Segment Message (SSM)	26	29	0	0
Beginning Of Message (BOM)	26	30	0	0
Continuation Of Message (COM)	26	30	0	0
End Of Message (EOM)	28	30	0	0

Table 4.2: number of instructions processed for Reassembly AAL5 and AAL3/4 messages (the DMA's clock has double speed of RISC's clock).

Clearly, the number of instructions required to process the AAL5 is less than that for AAL3/4. This has made the RISC core process more cells/sec for AAL5 than for AAL3/4. With the AAL5, there is no need to load and process the AAL header and AAL trailer. Table 4.3 shows the main difference between the processing of the ATM cells for both AAL3/4 and AAL 5 cells.

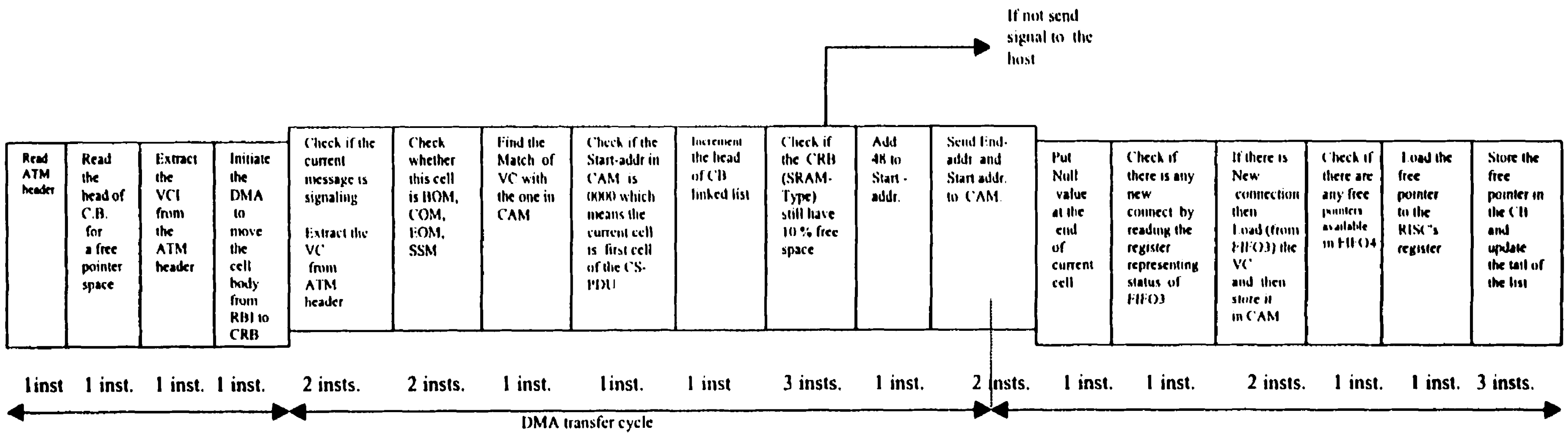
Operation	Instructions For AAL3/4	Instructions For AAL5
Read AAL header from RBI	1	0
Read AAL trailer from RBI	1	0
Mask VC (for AAL5) or VCI-MID (for AAL3/4) information needed for data matching	3	1
Mask only VCI for signaling test	0	1
Number of the comparison needed to recognize the PT of the current cell	4	2
Additional comparison needed to figure out the type of the current cell besides checking the Payload Type (PT)	0	1

Table 4.3: The main differences between the processing for AAL5 and AAL3/4 for ATM Reassembly

To make clear about how we get the results in Table 4.1 and Table 4.2 , it is important to describe the details of processing for every ATM cells. Figure 4.11 shows the total number of instructions that the RISC will process if the type of the incoming cell is BOM for AAL5. The RISC starts by loading the ATM header from RBI. Then it reads the head of the CB that contains the pointer space. The RISC initiates the DMA to move the cell body from the RBI to the pointer space inside the CRB. The highlighted area shows the instructions that the RISC can process during the data movement (the DMA's clock in this case has double RISC's clock). The DMA needs 12 RISC cycles (each cycle 32-bit) to finish transferring one ATM cell payload (48 bytes). During the data movements, the

RISC can execute instructions, such as finding a CAM match, updating the CAM entries, or calculating the available space inside the CB before the DMA finishes its job. After the DMA finishes its job, the RISC processor takes control of the bus. Then the RISC processes the linked list mechanism or reads the FIFO3 which contains new VC to be stored in CAM, and FIFO4 which contains a pointer to be stored in CB. Figure 4.12 shows the total instructions that RISC needs to process for one ATM cell type of COM. The number of instructions in the COM is the same as the BOM, but the COM has a slightly different mannerism than BOM. In the processing of the COM there is no need to update the CAM entries by the Start-address (this instruction exists in BOM processing). In the COM, processing is needed to update the previous cell's pointer (this instruction does not exist in BOM processing). Figure 4.13 shows the total instructions that are needed by the RISC processor to process EOM. The EOM has a higher number of instructions than BOM and COM because the RISC, at this point, needs to notify the host CPU that the PDU frame was received and is stored in the CRB. The notification to the CPU can be done by sending the Start-address of the location for the PDU frame inside the CRB and the VC that is related to this PDU frame to the FIFO 2. Figure 4.14 shows the total instructions that are needed by the RISC processor to process the SSM cell type. This message is a unique cell, which notifies that there are no more cells arriving for this VC. The RISC, after processing the SSM, sends the Start-address and VC host CPU through FIFO2. The SSM has the same mannerism as EOM, but the SSM shows fewer instructions than EOM because the SSM does not need any processing related to link-list mechanism and updating (i.e. reading the Start-address) the CAM.

Type of instructions that are processed by RISC in order to process BOM for ATM Reassembly for AAL5 where the FIFO3 and FIFO4 are not empty


If not send signal to the host
Read ATM header
Read the head of C.B. for a free pointer space
Extract the VCI from the ATM header
Initiate the DMA to move the cell body from RBI to CRB
Check if the current message is signaling Extract the VC from ATM header
Check whether this cell is BOM, COM, EOM, SSM
Find the Match of VC with the one in CAM
Check if the Start-addr in CAM is 0000 which means the current cell is first cell of the CS-PDU
Increment the head of CB linked list
Check if the CRB (SRAM-Type) still have 10 % free space
Add 48 to Start - addr.
Send End-addr and Start addr. to CAM.
Put Null value at the end of current cell
Check if there is any new connect by reading the register representing status of FIFO3
If there is New connection then Load (from FIFO3) the VC and then store it in CAM
Check if there are any free pointers available in FIFO4
Load the free pointer to the RISC's register
Store the free pointer in the CB and update the tail of the list
1inst
1 inst.
1 inst.
1 inst.
2 insts.
2 insts.
1 inst.
1inst.
1 inst
3 insts.
1 inst.
2 insts.
1 inst.
1 inst.
2 insts.
1 inst.
1 inst.
3 insts.
DMA transfer cycle

Processing needed before moving cell body = 4instructions

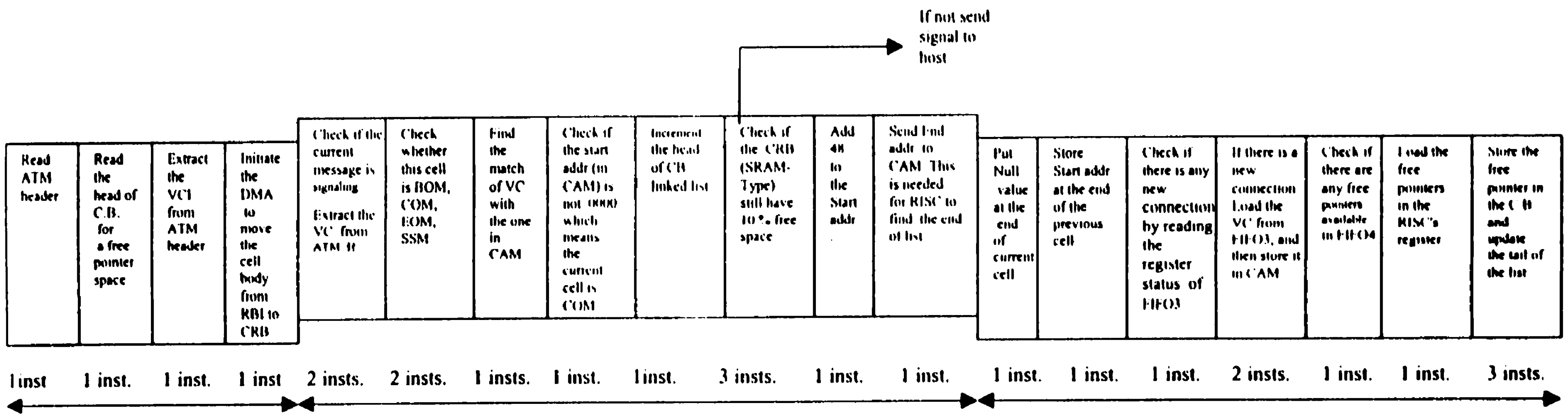
12 instruction that the RISC core can process. during the data movement.

Note: The DMA needs 11 RISC cycle to move 48-byte

10 insts are processed after moving the payload Some of the above insts. need to use the local bus to move data .

Figure 4.11:Total instruction for BOM of AAL5 is 26 inst.

ATM Reassembly for AAL5 processing scheme of COM
where the FIFO3 and FIFO4 are not empty


If not send signal to host
Read ATM header
Read the head of C.B. for a free pointer space
Extract the VCI from ATM header
Initiate the DMA to move the cell body from RBI to CRB
Check if the current message is signaling Extract the VC from ATM H
Check whether this cell is BOM, COM, EOM, SSM
Find the match of VC with the one in CAM
Check if the start addr (in CAM) is not 0000 which means the current cell is COM
Increment the head of CB linked list
Check if the CRB (SRAM-Type) still have 10 % free space
Add 48 to the Start addr
Send End addr to CAM This is needed for RISC to find the end of list
Put Null value at the end of current cell
Store Start addr at the end of the previous cell
Check if there is any new connection by reading the register status of FIFO3
If there is a new connection Load the VC from FIFO3, and then store it in CAM
Check if there are any free pointers available in FIFO4
Load the free pointers in the RISC's register
Store the free pointer in the CB and update the tail of the list
1 inst
1 inst.
1 inst.
1 inst
2 insts.
2 insts.
1 insts.
1 inst.
1 inst.
3 insts.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
2 insts.
1 inst.
1 inst.
3 insts.

Processing needed before moving cell body
→ 4 instruction

12 instructions the RISC core can process during the data movement

Note: The DMA needs 11 RISC cycle to move 48 byte
clock cycle

10 instructions are processed after moving the payload
Some of the above insts. need to use the local bus to move data .

Figure 4.12:Total instruction for COM of AAL5 is 26 inst.

ATM Reassembly for AAL5 processing scheme of EOM
where the FIFO3 and FIFO4 are not empty

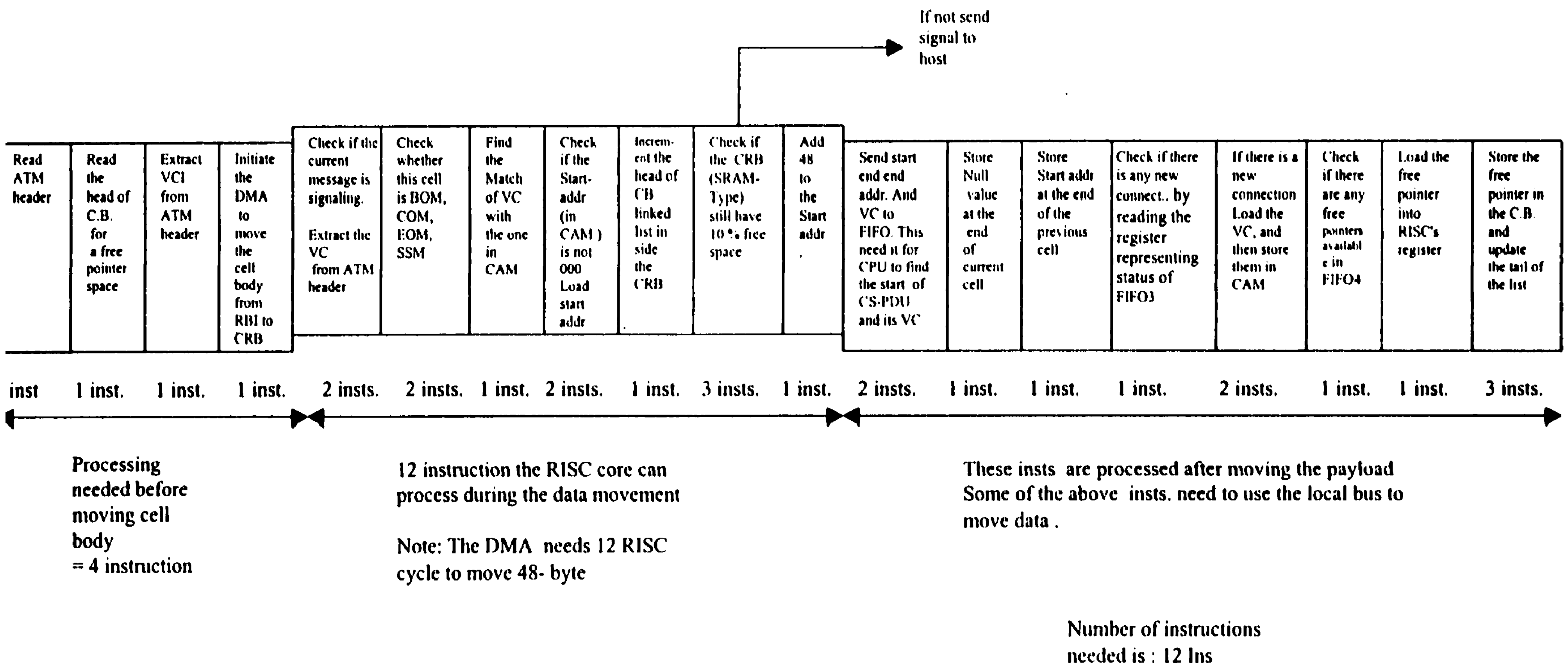

If not send signal to host
Read ATM header
Read the head of C.B. for a free pointer space
Extract VCI from ATM header
Initiate the DMA to move the cell body from RBI to CRB
Check if the current message is signaling. Extract the VC from ATM header
Check whether this cell is BOM, COM, EOM, SSM
Find the Match of VC with the one in CAM
Check if the Start-addr (in CAM) is not 000 Load start addr
Increm-ent the head of CB linked list in side the CRB
Check if the CRB (SRAM-Type) still have 10 % free space
Add 48 to the Start addr
Send start end end addr. And VC to FIFO. This need it for CPU to find the start of CS-PDU and its VC
Store Null value at the end of current cell
Store Start addr at the end of the previous cell
Check if there is any new connect.. by reading the register representing status of FIFO3
If there is a new connection Load the VC, and then store them in CAM
Check if there are any free pointers availabl e in FIFO4
Load the free pointer into RISC's register
Store the free pointer in the C.B. and update the tail of the list
inst
1 inst.
1 inst.
1 inst.
2 insts.
2 insts.
1 inst.
2 insts.
1 inst.
3 insts.
1 inst.
2 insts.
1 inst.
1 inst.
1 inst.
2 insts.
1 inst.
1 inst.
3 insts.
Processing needed before moving cell body = 4 instruction
12 instruction the RISC core can process during the data movement
Note: The DMA needs 12 RISC cycle to move 48- byte
These insts are processed after moving the payload Some of the above insts. need to use the local bus to move data .
Number of instructions needed is : 12 Ins

Figure 4.13: Total instruction for EOM of AAL5 is 28 inst.

ATM Reassembly for AAL5 processing scheme of SSM
where the FIFO3 and FIFO4 are not empty

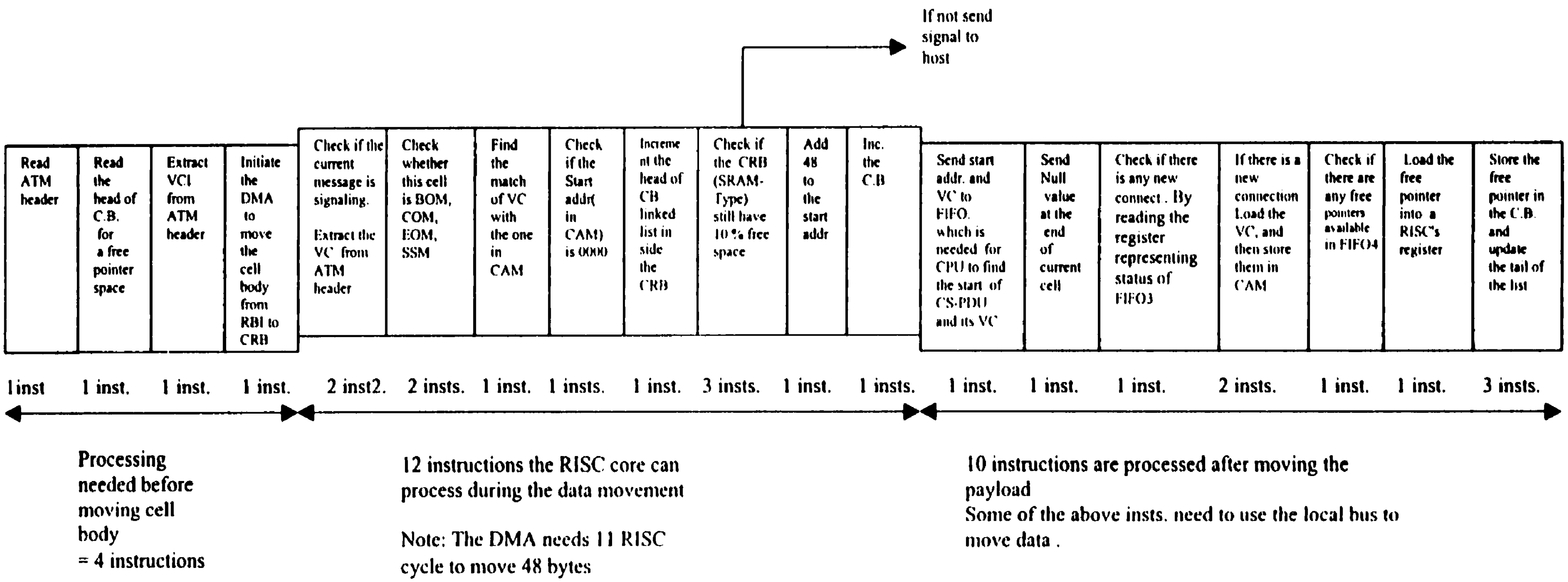

If not send signal to host
Read ATM header
Read the head of C.B. for a free pointer space
Extract VCI from ATM header
Initiate the DMA to move the cell body from RBI to CRB
Check if the current message is signaling. Extract the VC from ATM header
Check whether this cell is BOM, COM, EOM, SSM
Find the match of VC with the one in CAM
Check if the Start addr(in CAM) is 0000
Increme nt the head of CB linked list in side the CRB
Check if the CRB (SRAM-Type) still have 10 % free space
Add 48 to the start addr
Inc. the C.B
Send start addr. and VC to FIFO. which is needed for CPU to find the start of CS-PDU and its VC
Send Null value at the end of current cell
Check if there is any new connect.. By reading the register representing status of FIFO3
If there is a new connection Load the VC, and then store them in CAM
Check if there are any free pointers available in FIFO4
Load the free pointer into a RISC's register
Store the free pointer in the C.B. and update the tail of the list
1 inst
1 inst.
1 inst.
1 inst.
2 inst2.
2 insts.
1 inst.
1 insts.
1 inst.
3 insts.
1 inst.
1 insts.
1 inst.
1 inst.
1 inst.
2 insts.
1 inst.
1 inst.
3 insts.
Processing needed before moving cell body = 4 instructions
12 instructions the RISC core can process during the data movement
Note: The DMA needs 11 RISC cycle to move 48 bytes
10 instructions are processed after moving the payload
Some of the above insts. need to use the local bus to move data .

Figure 4.14: Total instruction for SSM of AAL5 is 26 inst.

Figure 4.15 shows thows the total amount and the type of the structions that the RISC processor needs to process the BOM for AAL3/4. The RISC starts reading the cell headers and trailer. These headers contain the information (i.e. VCI, MID and PT) that help the RISC find the match for VCI-MID and to recognize the type of incoming cell. This information will keep the RISC busy during the data movements which take 11 cycles (44 bytes, each cycle is 32-bit). The local bus is busy during the data movement so RISC does not access it. The RISC does not need the local bus during the execution of the instructions including: finding a CAM match or tracing the CB size. While the cell body has already been moved from RBI to the CRB, the RISC controls the local bus and is able to execute the linked list mechanism and can also read from FIFO3 and FIFO4. Figure 4.16 shows the instructions needed by the RISC processor to process one ATM cell type COM for AAL3/4. Figure 4.17 shows the total instructions that are needed to process EOM cell, the RISC with this type of message needs to send the VCI-MID and the Start-address to the FIFO2. There is no need to update the CAM entries. Figure 4.18 shows the total amount and the type of the instructions needed for ATM cell type SSM. The processing time needed is less for SSM than it is for BOM, COM and EOM where there is no need to update the CAM entries (Start- and End-address) or to process the linked list functions.

ATM Reassembly for AAL3/4 processing scheme of BOM
where the FIFO3 and FIFO4 are not empty

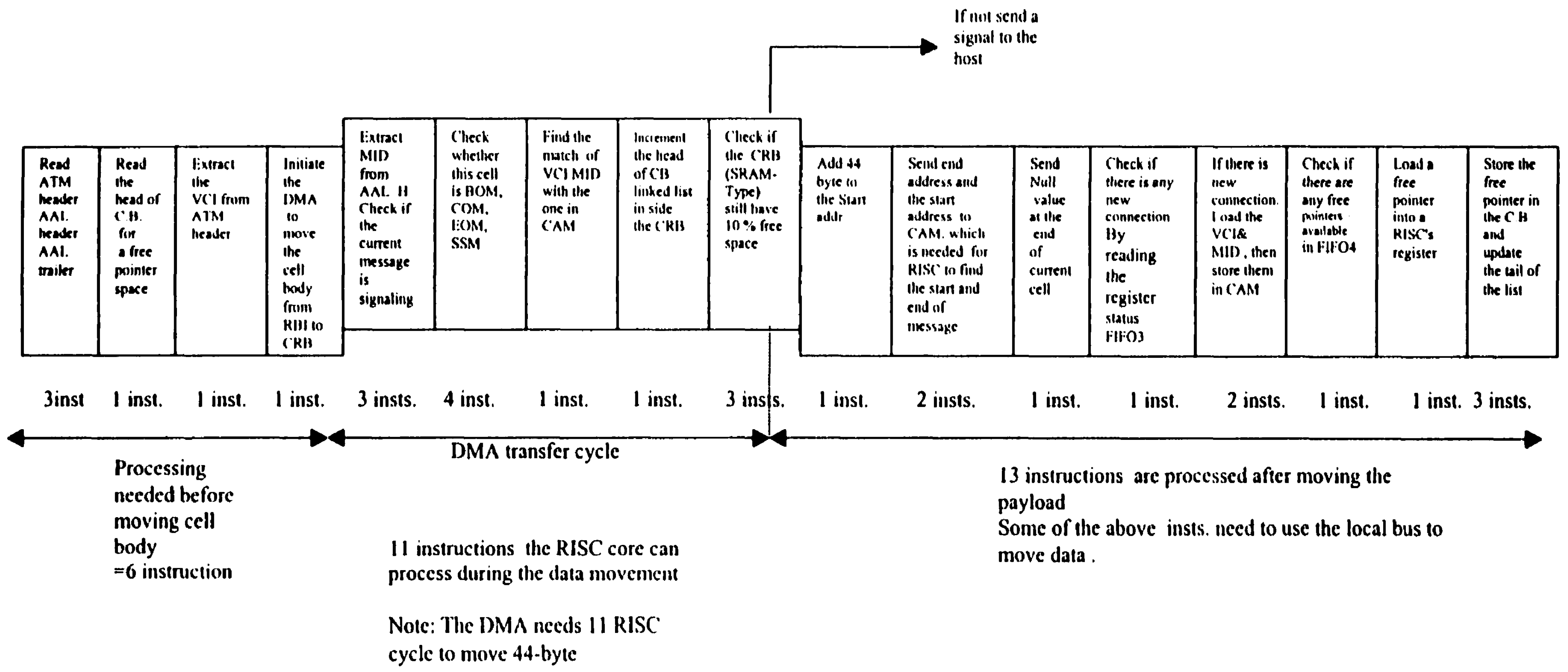

If not send a signal to the host
Read ATM header AAL header AAL trailer
Read the head of C.B. for a free pointer space
Extract the VCI from ATM header
Initiate the DMA to move the cell body from RBI to CRB
Extract MID from AAL H Check if the current message is signaling
Check whether this cell is BOM, COM, EOM, SSM
Find the match of VCI MID with the one in CAM
Increment the head of CB linked list in side the CRB
Check if the CRB (SRAM-Type) still have 10 % free space
Add 44 byte to the Start addr
Send end address and the start address to CAM, which is needed for RISC to find the start and end of message
Send Null value at the end of current cell
Check if there is any new connection By reading the register status FIFO3
If there is new connection. Load the VCI& MID , then store them in CAM
Check if there are any free pointers available in FIFO4
Load a free pointer into a RISC's register
Store the free pointer in the C.B and update the tail of the list
3inst
1 inst.
1 inst.
1 inst.
3 insts.
4 inst.
1 inst.
1 inst.
3 insts.
1 inst.
2 insts.
1 inst.
1 inst.
2 insts.
1 inst.
1 inst.
3 insts.
Processing needed before moving cell body =6 instruction
DMA transfer cycle
11 instructions the RISC core can process during the data movement
Note: The DMA needs 11 RISC cycle to move 44-byte
13 instructions are processed after moving the payload
Some of the above insts. need to use the local bus to move data .

Figure 4.15: Total instruction for BOM of AAL3/4 is 30 inst.

ATM Reassembly for AAL3/4 processing scheme of COM,
where the FIFO3 and FIFO4 are not empty

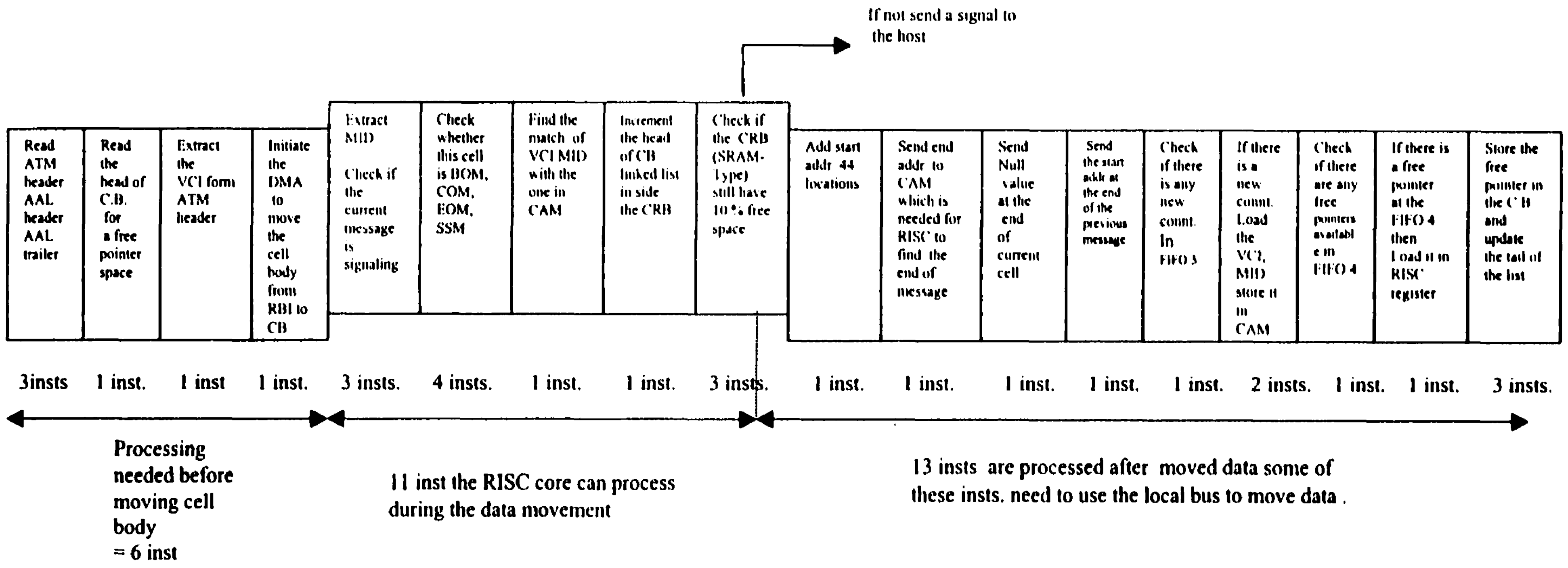
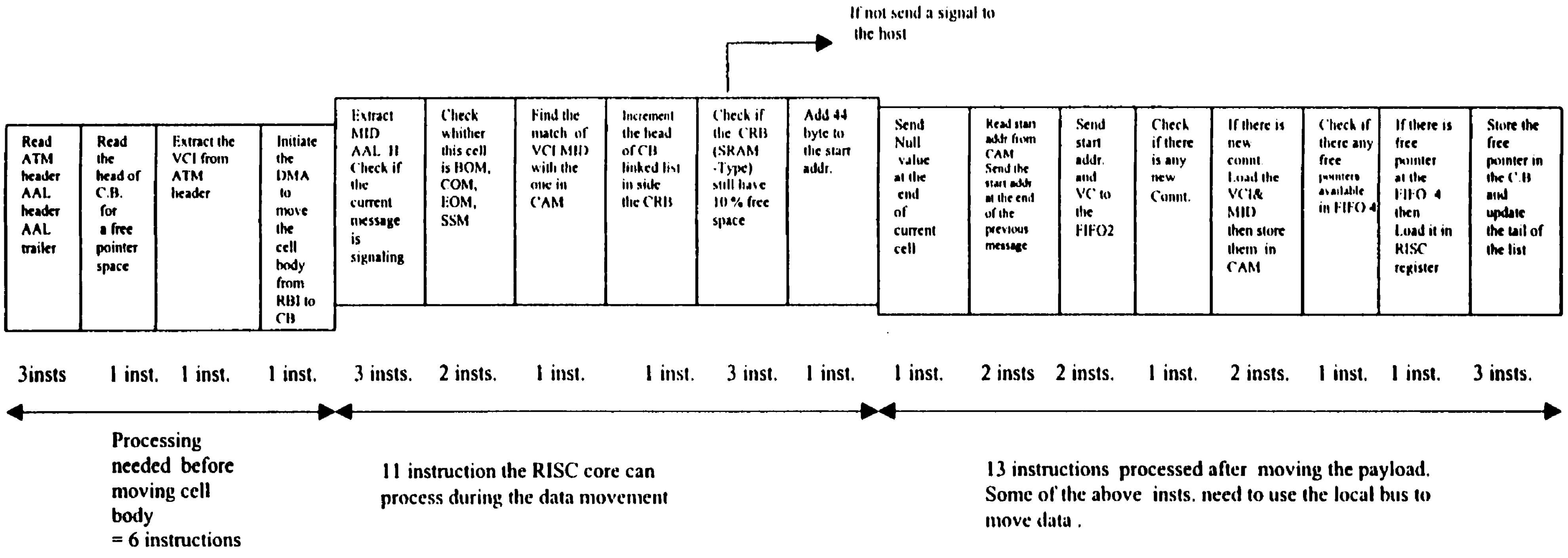

If not send a signal to the host
Read ATM header AAL header AAL trailer
Read the head of C.B. for a free pointer space
Extract the VCI form ATM header
Initiate the DMA to move the cell body from RBI to CB
Extract MID Check if the current message is signaling
Check whether this cell is BOM, COM, EOM, SSM
Find the match of VCI MID with the one in CAM
Increment the head of CB linked list in side the CRB
Check if the CRB (SRAM-Type) still have 10 % free space
Add start addr 44 locations
Send end addr to CAM which is needed for RISC to find the end of message
Send Null value at the end of current cell
Send the start addr at the end of the previous message
Check if there is any new connt. In FIFO 3
If there is a new connt. Load the VCI, MID store it in CAM
Check if there are any free pointers availabl e in FIFO 4
If there is a free pointer at the FIFO 4 then Load it in RISC register
Store the free pointer in the CB and update the tail of the list
3insts
1 inst.
1 inst
1 inst.
3 insts.
4 insts.
1 inst.
1 inst.
3 insts.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
2 insts.
1 inst.
1 inst.
3 insts.
Processing needed before moving cell body = 6 inst
11 inst the RISC core can process during the data movement
13 insts are processed after moved data some of these insts. need to use the local bus to move data .

Figure 4.16: Total instruction for COM of AAL3/4 is 30 inst.

ATM Reassembly for AAL3/4 processing scheme of EOM
where the FIFO3 and FIFO4 are not empty



If not send a signal to the host
Read ATM header AAL header AAL trailer
Read the head of C.B. for a free pointer space
Extract the VCI from ATM header
Initiate the DMA to move the cell body from RBI to CB
Extract MID AAL II Check if the current message is signaling
Check whither this cell is BOM, COM, EOM, SSM
Find the match of VCI MID with the one in CAM
Increment the head of CB linked list in side the CRB
Check if the CRB (SRAM -Type) still have 10 % free space
Add 44 byte to the start addr.
Send Null value at the end of current cell
Read start addr from CAM Send the start addr at the end of the previous message
Send start addr. and VC to the FIFO2
Check if there is any new Connt.
If there is new connt. Load the VCI& MID then store them in CAM
Check if there any free pointers available in FIFO 4
If there is free pointer at the FIFO 4 then Load it in RISC register
Store the free pointer in the C.B and update the tail of the list
3insts
1 inst.
1 inst.
1 inst.
3 insts.
2 insts.
1 inst.
1 inst.
3 inst.
1 inst.
1 inst.
2 insts
2 insts.
1 inst.
2 insts.
1 inst.
1 inst.
3 insts.
Processing needed before moving cell body = 6 instructions
11 instruction the RISC core can process during the data movement
13 instructions processed after moving the payload. Some of the above insts. need to use the local bus to move data .

Figure 4.17: Total instruction for EOM of AAL3/4 is 30 inst.

ATM Reassembly for AAL3/4 processing scheme of SSM, where the FIFO3 and FIFO4 are not empty

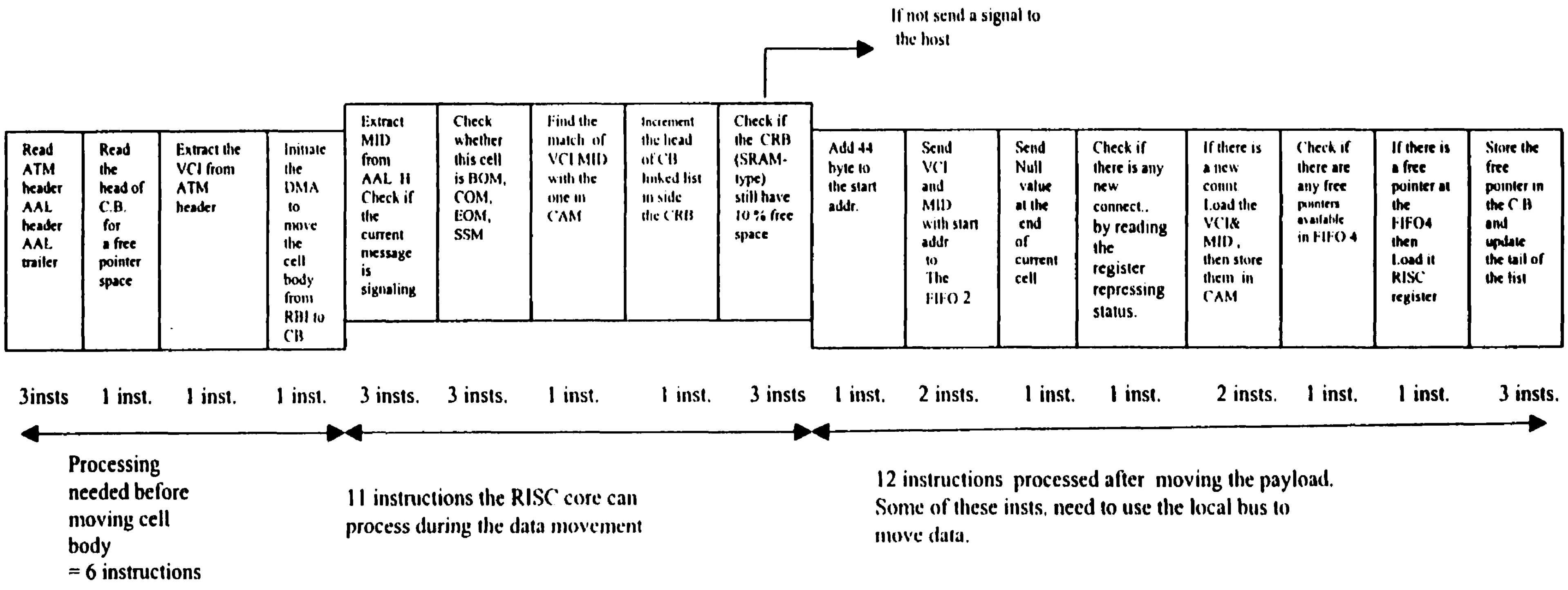

If not send a signal to the host
Read ATM header AAL header AAL trailer
Read the head of C.B. for a free pointer space
Extract the VCI from ATM header
Initiate the DMA to move the cell body from RBI to CB
Extract MID from AAL II Check if the current message is signaling
Check whether this cell is BOM, COM, EOM, SSM
Find the match of VCI MID with the one in CAM
Increment the head of CB linked list in side the CRB
Check if the CRB (SRAM-type) still have 10 % free space
Add 44 byte to the start addr.
Send VCI and MID with start addr to The FIFO 2
Send Null value at the end of current cell
Check if there is any new connect.. by reading the register repressing status.
If there is a new conn. Load the VCI& MID , then store them in CAM
Check if there are any free pointers available in FIFO 4
If there is a free pointer at the FIFO4 then Load it RISC register
Store the free pointer in the C.B and update the tail of the list
3insts
1 inst.
1 inst.
1 inst.
3 insts.
3 insts.
1 inst.
1 inst.
3 insts
1 inst.
2 insts.
1 inst.
1 inst.
2 insts.
1 inst.
1 inst.
3 insts.
Processing needed before moving cell body = 6 instructions
11 instructions the RISC core can process during the data movement
12 instructions processed after moving the payload. Some of these insts. need to use the local bus to move data.

Figure 4.18: Total instruction for SSM of AAL3/4 is 29 inst.

4.3.2 Segmentation Function

Segmentation function simulation using the DMA's clock at the same speed as the RISC 's clock means the DMA needs 22 cycles to move one cell body from CSB to SBI for AAL3/4, where it needs 24 cycle for AAL5. While moving data, the DMA controls the local bus to move the cell body from CSB to the SBI. The RISC also needed to send the ATM cell header (if the transfer cell is AAL5 type) or send the ATM header, the AAL header and the AAL trailer (if the transfer cell is AAL3/4 type) from RISC's register to the SBI. In this case, the RISC has to wait until the DMA completes the data movement. Then, the RISC is able to control the local bus and transfer the data register to the SBI. The total instructions that were needed for Segmentation function, where the DMA's clock is the same RISC's clock cycle are shown in Table 4.4. The RISC has several idle cycles during the data movements and obviously is not able to send any of its data registers on the local bus.

Type of cells for For AAL3/4 and AAL5	**Total instructions**		**No. of idle cycls**	
	AAL5	**AAL3/4**	**AAL5**	**AAL3/4**
Single Segment Message (SSM)	26	26	17	13
Beginning Of Message (BOM)	26	26	18	14
Continuation Of Message (COM)	26	26	22	18
End Of Message (EOM)	26	26	21	17

Table 4.4: Number of RISC instructions processed and the idle cycles for Segmentation messages (the DMA have the same clock cycle as the RISC)

The number of idle cycles is quite high and the RISC wastes almost 70 % of its power. We tried to eliminate the number of idle cycles by increasing the DMA's clock speed to be double that of the RISC's, thus forcing the DMA to finish its job by moving the cell body within 11 instructions for AAL3/4 and 12 instruction for AAL5 (Table 4.5).

Type of cells	**Total instructions**		**No. of idle cycles**	
For AAL 5 and AAL3/4	**AAL5**	**AAL3/4**	**AAL5**	**AAL3/4**
Single Segment Message (SSM)	14	15	5	2
Beginning Of Message (BOM)	14	15	6	3
Continuation Of Message (COM)	14	15	10	7
End Of Message (EOM)	14	15	9	6

Table 4.5: Number of RISC instructions processed and the idle cycles for Segmentation messages (the DMA have the clock cycle double the RISC)

The performance of the processing gets better, but there is a loss of about 40% the RISC's power. We tried to improve our processing performance by pushing the DMA's clock to be triple that of the RISC core's clock (Table 4.6).

Type of cells **For AAL5 and AAL3/4**	**Total instructions**		**No. Of idle cycles**	
	AAL5	**AAL3/4**	**AAL5**	**AAL3/4**
Single Segment Message (SSM)	10	13	1	0
Beginning Of Message (BOM)	10	12	2	0
Continuation Of Message (COM)	10	11	4	2
End Of Message (EOM)	10	11	5	3

Table 4.6: Number of instructions processed and the Idle instructions for Segmentation messages where the DMA has triple the clock cycle of the RISC

For the Segmentation part of the network interface, we found that using a DMA controller faster than the RISC core will improve the performance. Because the RISC core can perform little processing while the DMA controller is moving the payload from the Cell Segmentation Buffer (CSB) to the Send Buffer Interface (SBI), the RISC core is forced to be idle for a few cycles until the DMA completes the payload transfer. Therefore, using a faster DMA will help to eliminate all idle cycles of the RISC core. The differences in the instructions that executed by AAL3/4 and AAL5 are shown in Table 4.7.

Operation	Instruction for AAL3/4	AAL5
Send AAL header from CSB to SBI	1	0
Send AAL trailer from CSB to SBI	1	0
Change the Sequence Number (SN) for each leaving cell	1*	0
Calculate the length of the AAL trailer	1*	0
Generate the AAL header	2*	0
* *Instruction executed during the moving data.*		

Table 4.7: The main differences between the AAL5 and AAL3/4 for ATM Segmentation

To make clear how we get the results in Table 4.4, Table 4.5 and Table 4.6, it is important to describe the processing details. The focus is given on the case where the DMA's clock runs at triple the clock cycle of the RISC. When the host decides to send a block of data to the other ATM host, the host CPU sends the CPCS-PDU frame to the CSB. The host CPU also sends all the information needed to transmit this frame through FIFO5, including VCI, VPI, and the location of the frame inside the CSB. Figure 4.19 shows the number of instructions required for ATM Segmentation function for BOM for AAL5. In order to generate the ATM header and calculate the PDU size, the RISC starts by reading all the information related to the PDU frame.

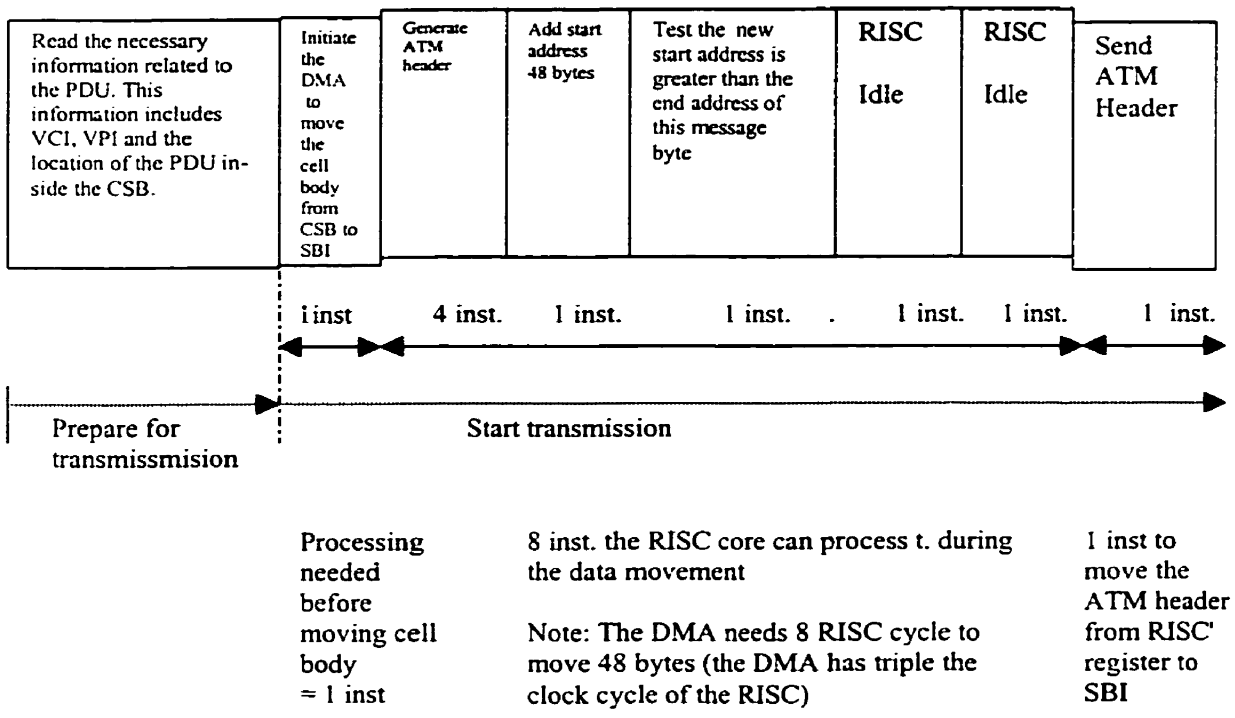

Read the necessary information related to the PDU. This information includes VCI, VPI and the location of the PDU inside the CSB.
Initiate the DMA to move the cell body from CSB to SBI
Generate ATM header
Add start address 48 bytes
Test the new start address is greater than the end address of this message byte
RISC Idle
RISC Idle
Send ATM Header
1 inst
4 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
Prepare for transmissmision
Start transmission
Processing needed before moving cell body = 1 inst
8 inst. the RISC core can process t. during the data movement
Note: The DMA needs 8 RISC cycle to move 48 bytes (the DMA has triple the clock cycle of the RISC)
1 inst to move the ATM header from RISC' register to SBI

Figure 4.19: ATM Segmentation for AAL5 processing scheme of BOM (Total instructions by BOM for AAL5 is 10 inst)

The RISC initiates the DMA to move the 48 bytes from the CSB to the SBI. The highlighted boxes show the RISC instructions that are executed during the data movement. These instructions include the generating of the ATM header and checking if there are more ATM cells to be sent for the same VC. After the DMA finishes its job by sending 48 bytes from the CSB to the SBI which needs 8 RISC cycle (the DMA's clock in this case has triple RISC's clock), the RISC sends the ATM header to the SBI. Figure 4.20 shows the total instructions needed by COM for AAL5. The RISC will be idle several cycles while waiting for the DMA to finish. It is clear that the RISC has more idle cycles in COM than BOM because, in this stage, the RISC has no need to generate the ATM header again (it already exists in the RISC's register).

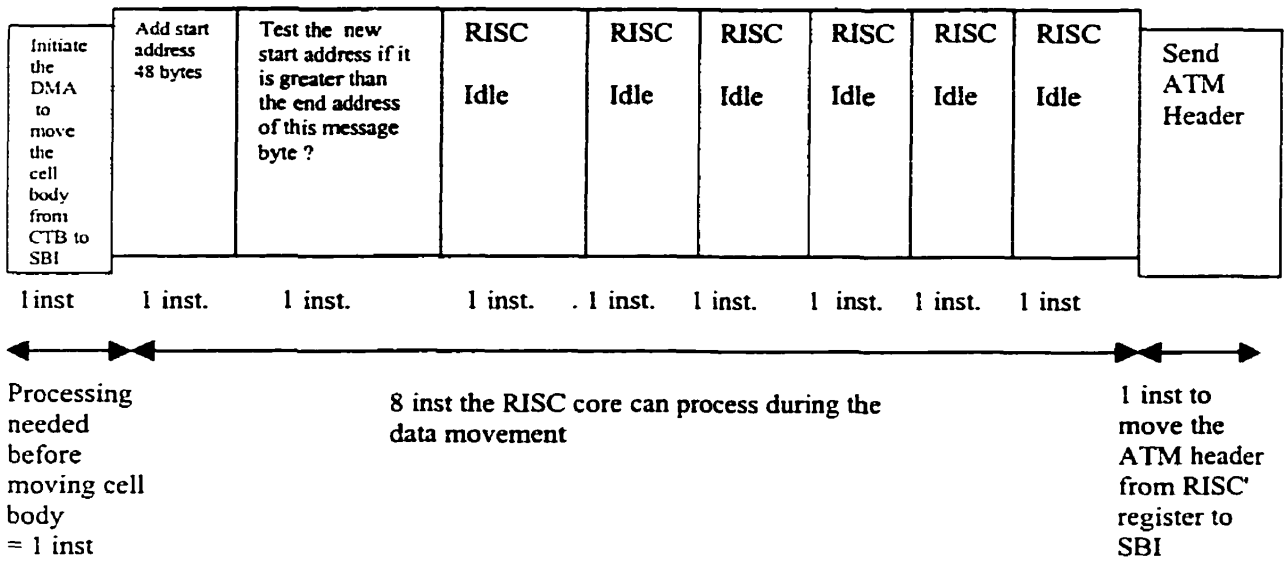

Initiate the DMA to move the cell body from CTB to SBI
Add start address 48 bytes
Test the new start address if it is greater than the end address of this message byte ?
RISC Idle
RISC Idle
RISC Idle
RISC Idle
RISC Idle
RISC Idle
Send ATM Header
1 inst
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst
Processing needed before moving cell body = 1 inst
8 inst the RISC core can process during the data movement
1 inst to move the ATM header from RISC' register to SBI

Figure 4.20: ATM Segmentation for AAL5 processing scheme of COM (Total instructions by COM for AAL5 is 10 inst)

Figure 4.21 shows the total instructions needed by EOM for AAL5 Segmentation function. The RISC has one more instruction than the COM, because the RISC needs to change the PT in ATM header from '0' to '1,' thus indicating that this cell is the last cell of the PDU frame.

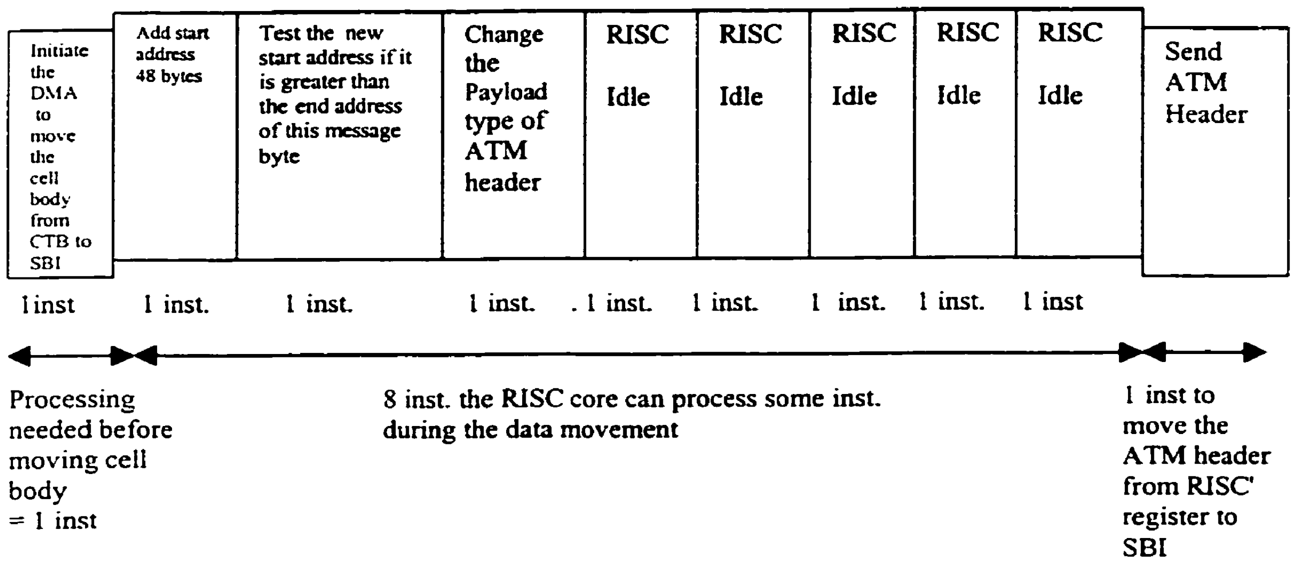

Initiate the DMA to move the cell body from CTB to SBI
Add start address 48 bytes
Test the new start address if it is greater than the end address of this message byte
Change the Payload type of ATM header
RISC Idle
RISC Idle
RISC Idle
RISC Idle
RISC Idle
Send ATM Header
1 inst
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst
Processing needed before moving cell body = 1 inst
8 inst. the RISC core can process some inst. during the data movement
1 inst to move the ATM header from RISC' register to SBI

Figure 4.21: ATM Segmentation for AAL5 processing scheme of EOM (Total instructions by EOM for AAL5 is 10 inst)

Figure 4.22 shows the total instruction by SSM for AAL5. The RISC was busy during the data movement and there is one idle cycle shown because the RISC was finsheing its job before the DMA finishing its task.

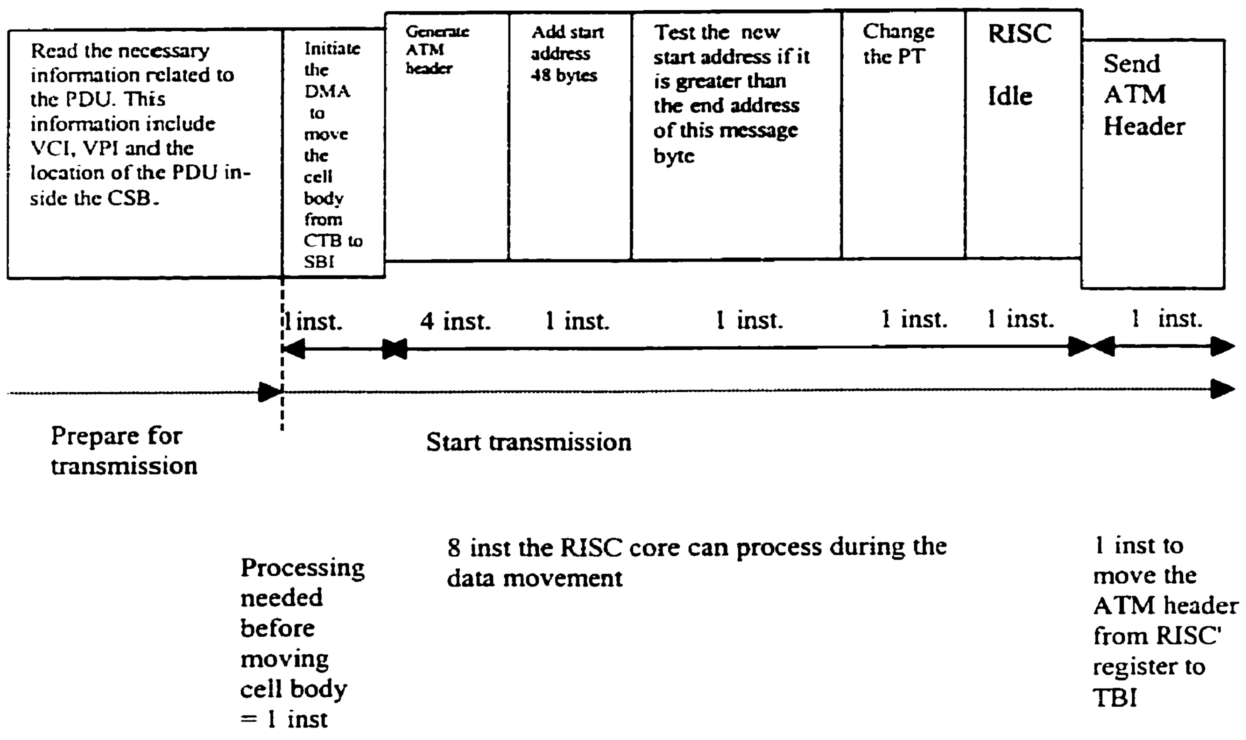

Read the necessary information related to the PDU. This information include VCI, VPI and the location of the PDU in-side the CSB.
Initiate the DMA to move the cell body from CTB to SBI
Generate ATM header
Add start address 48 bytes
Test the new start address if it is greater than the end address of this message byte
Change the PT
RISC Idle
Send ATM Header
1 inst.
4 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
Prepare for transmission
Start transmission
Processing needed before moving cell body = 1 inst
8 inst the RISC core can process during the data movement
1 inst to move the ATM header from RISC' register to TBI

Figure 4.22: ATM Segmentation for AAL5 processing scheme of SSM (Total instructions by SSM for AAL5 is 10 inst)

Figure 4.23 shows the total instruction that the RISC needs to process the Segmentation function for BOM cell type for AAL3/4. After reading the necessary information from FIFO5, the RISC starts to generate the ATM header and AAL header based on the information that the RISC reads from FIFO5. After the DMA finishes transferring the cell body from the CSB to SBI, the RISC starts sending the cell headers and trailer to the SBI.

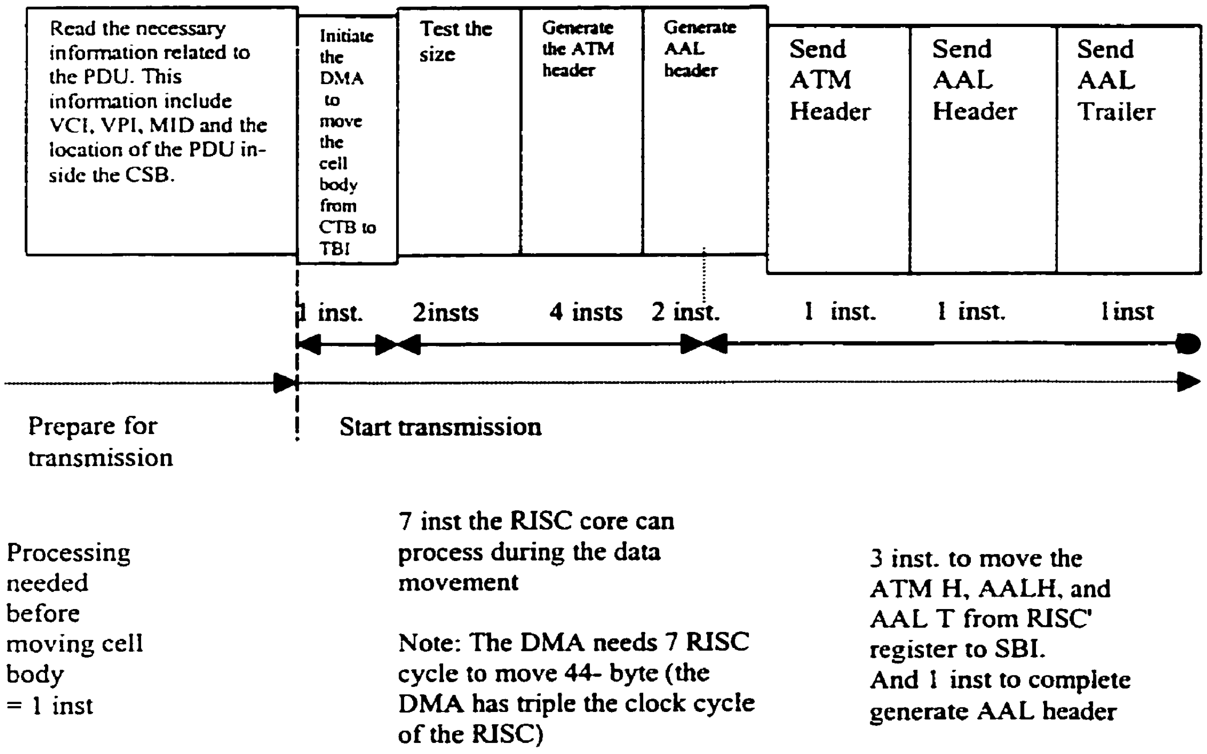

Read the necessary information related to the PDU. This information include VCI, VPI, MID and the location of the PDU in-side the CSB.
Initiate the DMA to move the cell body from CTB to TBI
Test the size
Generate the ATM header
Generate AAL header
Send ATM Header
Send AAL Header
Send AAL Trailer
1 inst.
2insts
4 insts
2 inst.
1 inst.
1 inst.
1inst
Prepare for transmission
Start transmission
Processing needed before moving cell body = 1 inst
7 inst the RISC core can process during the data movement
Note: The DMA needs 7 RISC cycle to move 44- byte (the DMA has triple the clock cycle of the RISC)
3 inst. to move the ATM H, AALH, and AAL T from RISC' register to SBI.
And 1 inst to complete generate AAL header

Figure 4.23: ATM Segmentation for AAL3/4 processing scheme of BOM (Total instructions by BOM for AAL3/4 is 12 inst.)

Figure 4.24 shows the total instruction that the RISC needs by COM for AAL3/4. The idle cycles are more than the BOM because the RISC in this stage has no need to generate the ATM header, but the RISC needs to change the PT and SN inside the AAL header from '10' to '00.'

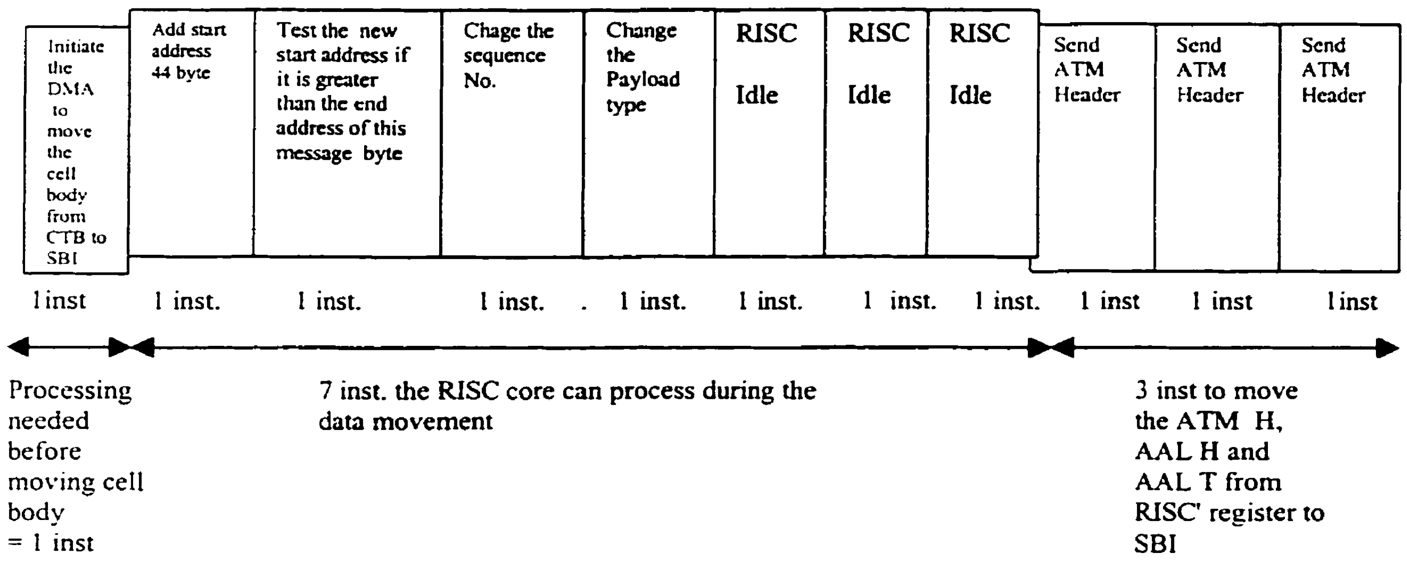

Initiate the DMA to move the cell body from CTB to SBI
Add start address 44 byte
Test the new start address if it is greater than the end address of this message byte
Chage the sequence No.
Change the Payload type
RISC Idle
RISC Idle
RISC Idle
Send ATM Header
Send ATM Header
Send ATM Header
1 inst
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst
1 inst
1 inst
Processing needed before moving cell body = 1 inst
7 inst. the RISC core can process during the data movement
3 inst to move the ATM H, AAL H and AAL T from RISC' register to SBI

Figure 4.24: ATM Segmentation for AAL3/4 processing scheme of COM (instructions by COM for AAL3/4 is 11 inst)

Figure 4.25 shows the total instruction that RISC needs for Segmentation function by EOM for AAL3/4. The RISC in this stage needs to calculate the actual data size inside the cell body (BOM and COM the cell bodies are fixed 44 bytes (ITUT93)).

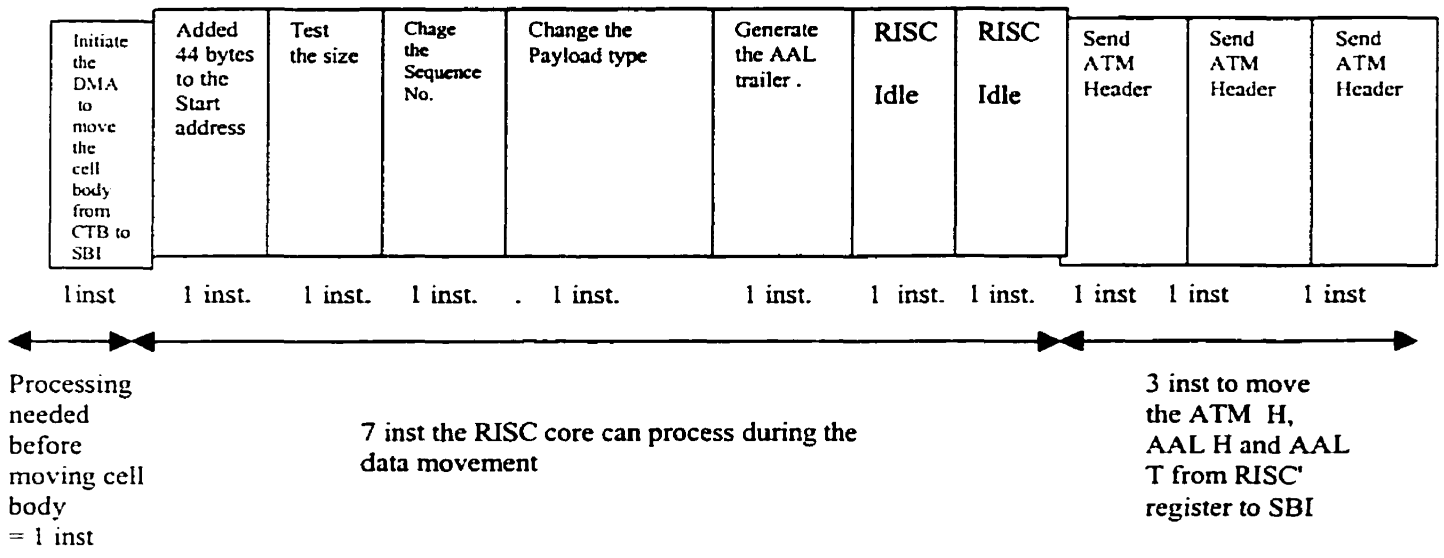

Initiate the DMA to move the cell body from CTB to SBI
Added 44 bytes to the Start address
Test the size
Chage the Sequence No.
Change the Payload type
Generate the AAL trailer .
RISC Idle
RISC Idle
Send ATM Header
Send ATM Header
Send ATM Header
1 inst
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst.
1 inst
1 inst
1 inst
Processing needed before moving cell body = 1 inst
7 inst the RISC core can process during the data movement
3 inst to move the ATM H, AAL H and AAL T from RISC' register to SBI

Figure 4.25: ATM Segmentation for AAL3/4 processing scheme of EOM (Total instructions by EOM for AAL3/4 is 12 inst)

Figure 4.26 shows the total instruction by SSM for AAL3/4. The RISC was busy during the data movement and there was no idle cycle shown because the RISC was busy generating the cell header and trailer.

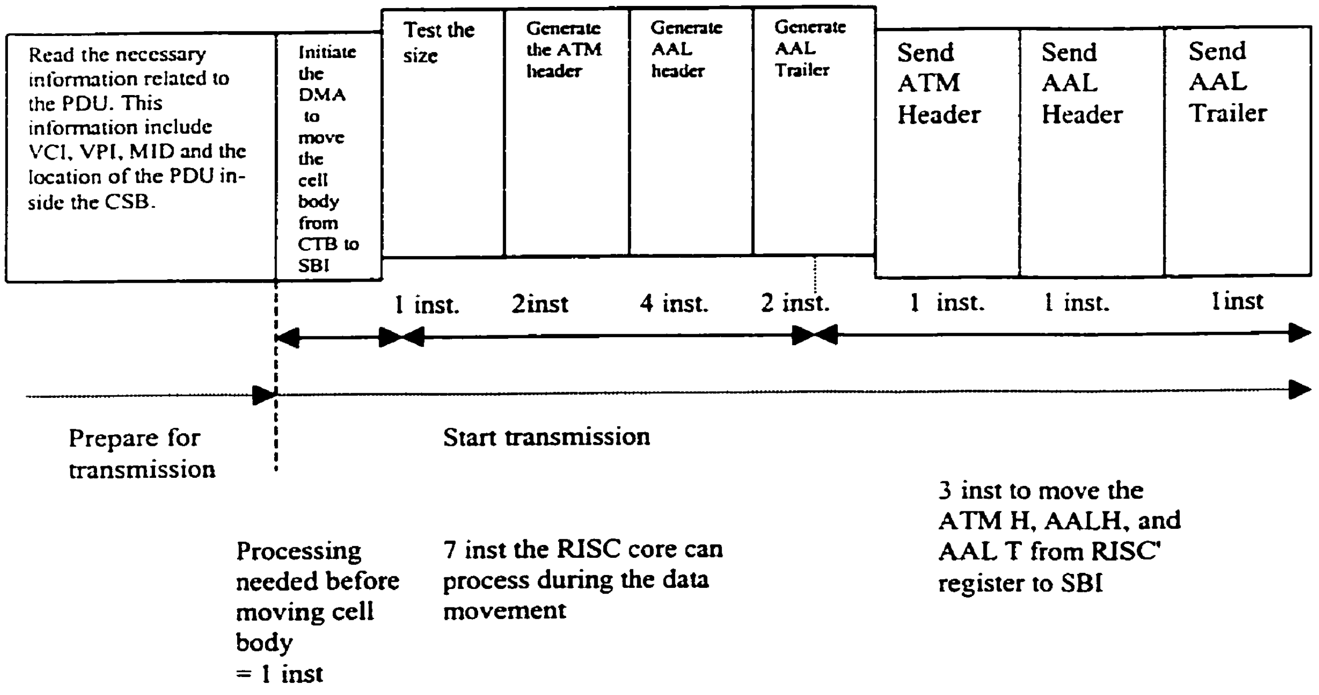
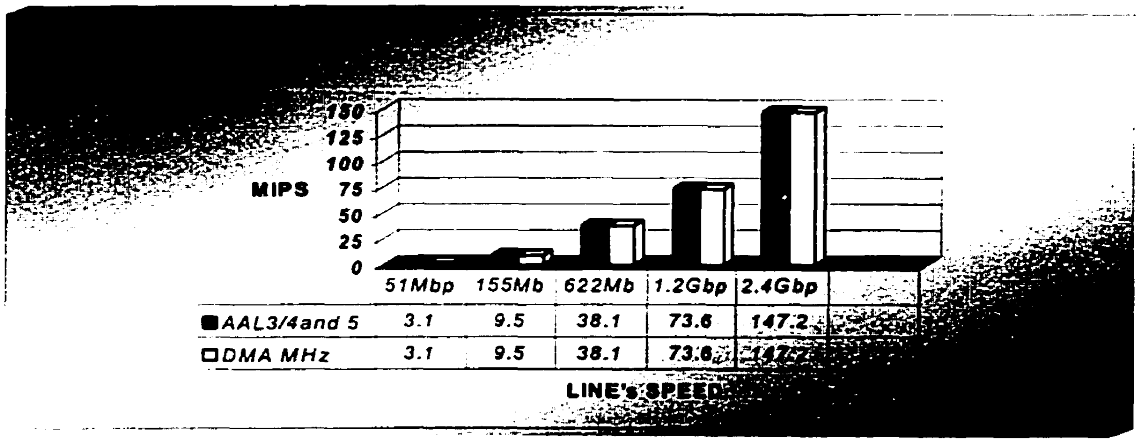

Read the necessary information related to the PDU. This information include VCI, VPI, MID and the location of the PDU inside the CSB.
Initiate the DMA to move the cell body from CTB to SBI
Test the size
Generate the ATM header
Generate AAL header
Generate AAL Trailer
Send ATM Header
Send AAL Header
Send AAL Trailer
1 inst.
2inst
4 inst.
2 inst.
1 inst.
1 inst.
1inst
Prepare for transmission
Start transmission
3 inst to move the ATM H, AALH, and AAL T from RISC' register to SBI
Processing needed before moving cell body = 1 inst
7 inst the RISC core can process during the data movement

Figure 4.26: ATM Segmentation for AAL3/4 processing scheme of SSM (Total instructions by SSM for AAL3/4 is 13 inst)

4.4 NI Performance Evaluation

The VHDL simulator gives more details and more accurate results than the SPIM Simulator. We have completed this simulator for AAL3/4 and AAL5 using Xilinx FPGA tool version 1.5. The reason for using Xilinx framework is that it provides a good environment for simulation and testing the VHDL model. Our intention was not to use FPGA as a target chip for our future implementation.

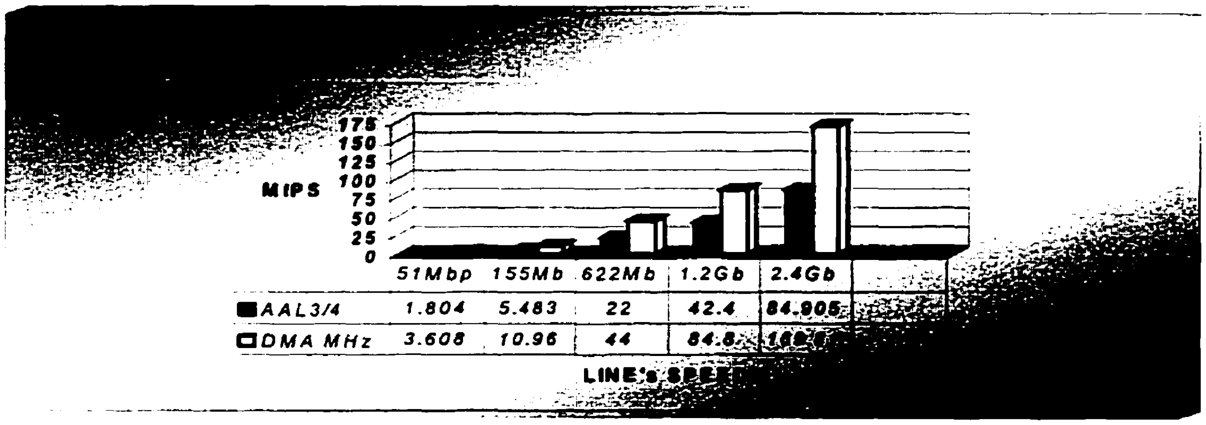
We have used the NI simulation to measure the amount of processing required for different transmission line speeds. For the Segmentation section of the network interface, a RISC core supported by a DMA having the same clock as the RISC can support 1.2Gbps transmission lines for AAL3/4 and AAL5 with 74 MHz, where a 147 MHz is required to support 2.4 Gbps line (Figure 4.27).



	51Mbp	155Mb	622Mb	1.2Gbp	2.4Gbp
■ AAL3/4and 5	3.1	9.5	38.1	73.6	147.2
□ DMA MHz	3.1	9.5	38.1	73.6	147.2

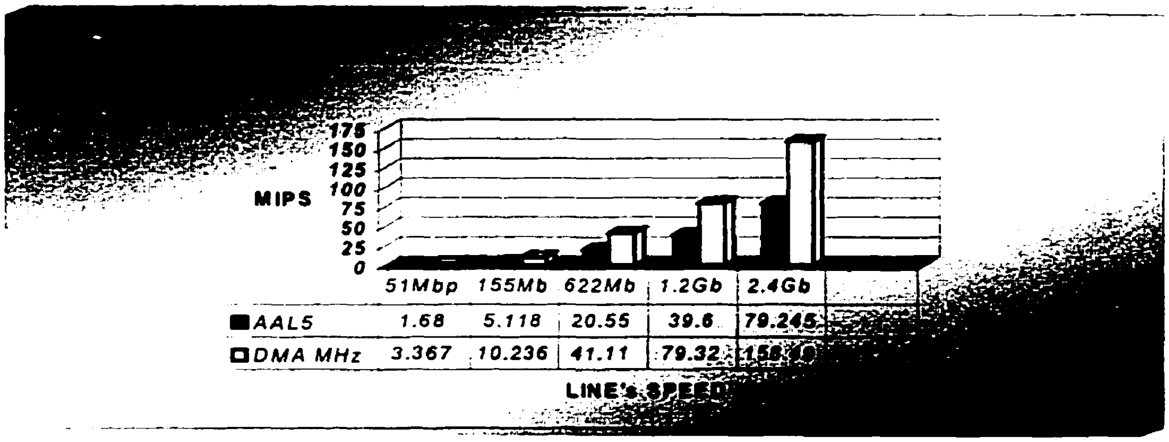
Figure 4.27: ATM Segmentation for AAL3/4 and AAL5 using DMA for data movement (the DMA has the same RISC's clock rate)

Figures 4.28 and 4.29 show the RISC processing speed for the AAL5 and AAL3/4 for segmentation function when the DMA has double clock speed than the previous rate.



	51Mbp	155Mb	622Mb	1.2Gb	2.4Gb
■ AAL3/4	1.804	5.483	22	42.4	84.805
□ DMA MHz	3.608	10.96	44	84.8	[illegible]

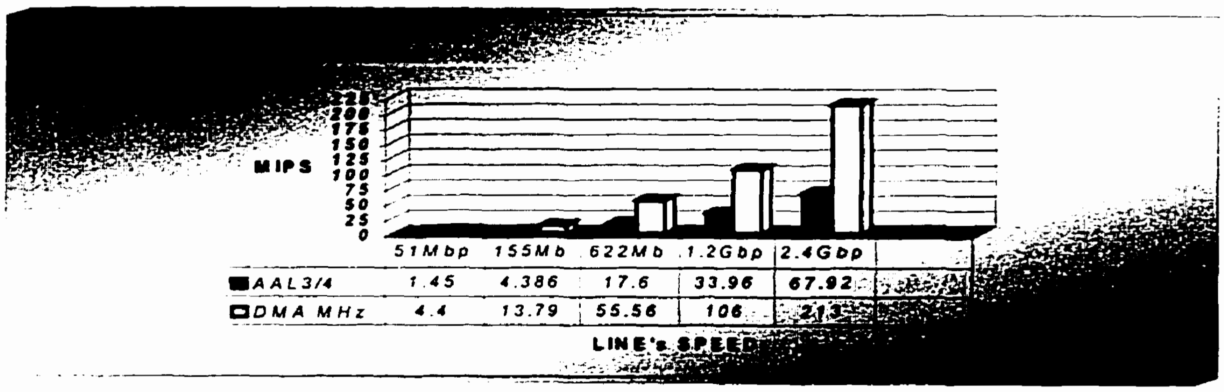
Figure 4.28: ATM Segmentation for AAL3/4 using DMA for data movement (the DMA has double RISC's clock rate)



	51Mbp	155Mb	622Mb	1.2Gb	2.4Gb
■ AAL5	1.68	5.118	20.55	39.6	79.245
□ DMA MHz	3.367	10.236	41.11	79.32	[illegible]

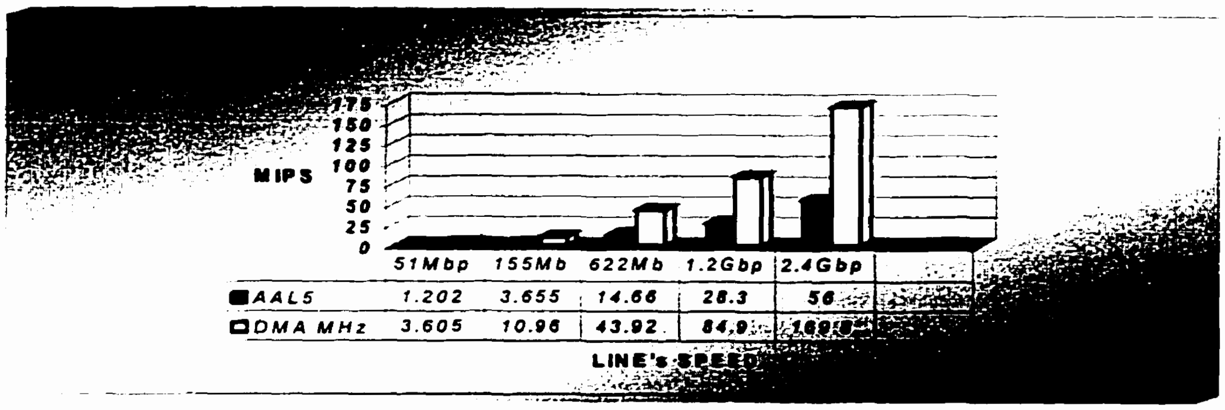
Figure 4.29: ATM Segmentation for AAL5 using DMA for data movement (the DMA has double RISC's clock rate)

Figures 4.30 and 4.31 show the RISC processing rate that is needed for ATM Segmentation for AAL 3/4 and AAL5, when the DMA has triple speed than the RISC core. Clearly, as the DMA getting faster, the RISC core will not need to be waiting, i.e., executing no operation instructions, while the NI local bus is busy due to the DMA moving data operation.



	51Mbp	155Mb	622Mb	1.2Gbp	2.4Gbp
■ AAL3/4	1.45	4.386	17.6	33.96	67.92
□ DMA MHz	4.4	13.79	55.56	106	213

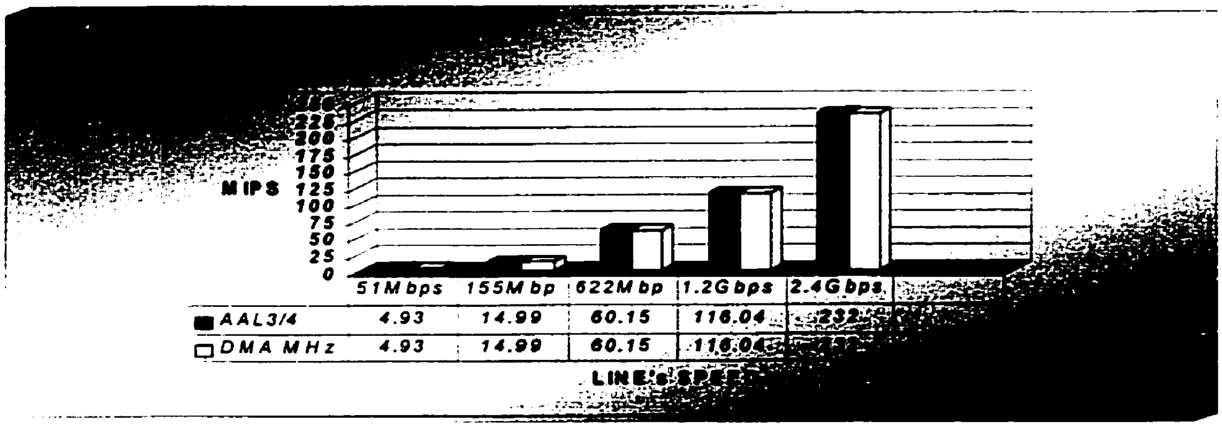
Figure 4.30: ATM Segmentation for AAL3/4 using DMA for data movement (the DMA has triple than the RISC's clock rate)



	51Mbp	155Mb	622Mb	1.2Gbp	2.4Gbp
■ AAL5	1.202	3.655	14.66	28.3	56
□ DMA MHz	3.605	10.96	43.92	84.9	169.8

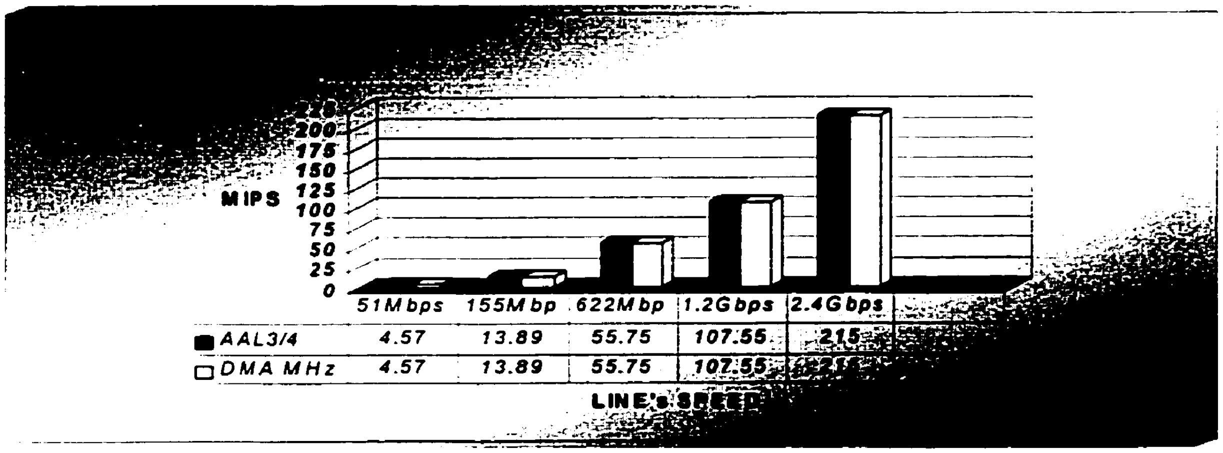
Figure 4.31: ATM Segmentation for AAL5 using DMA for data movement (the DMA has triple RISC's clock rate)

Figures 4.32 and 4.33 will present the RISC processing rate needed in order to process the ATM Reassembly for both functions AAL3/4 and AAL5, when the DMA has the same clock rate as the RISC processor.



	51Mbps	155Mbp	622Mbp	1.2Gbps	2.4Gbps
■ AAL3/4	4.93	14.99	60.15	116.04	232
□ DMA MHz	4.93	14.99	60.15	116.04	[illegible]

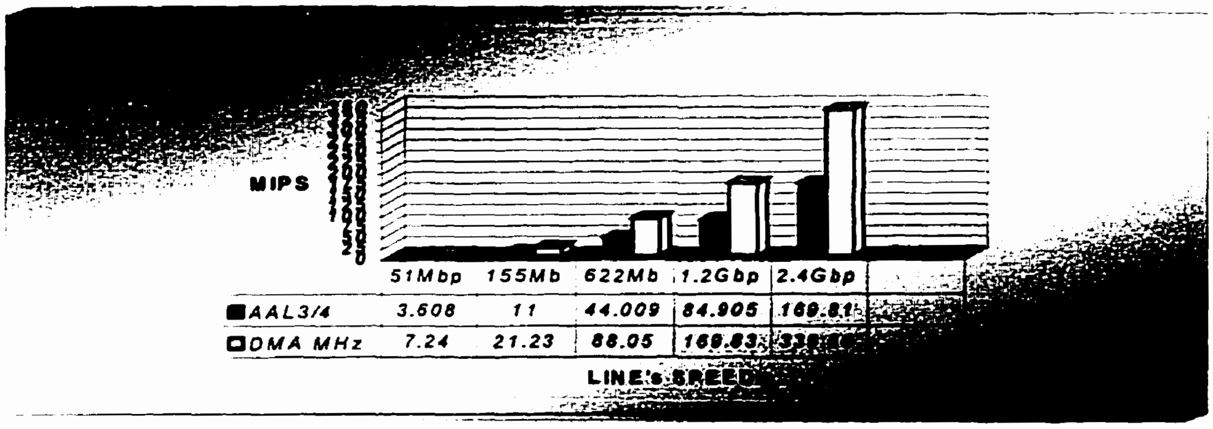
Figure 4.32: ATM Reassembly for AAL3/4 using DMA for data movement (the DMA has the same RISC's clock rate)



	51Mbps	155Mbp	622Mbp	1.2Gbps	2.4Gbps
■ AAL3/4	4.57	13.89	55.75	107.55	215
□ DMA MHz	4.57	13.89	55.75	107.55	215

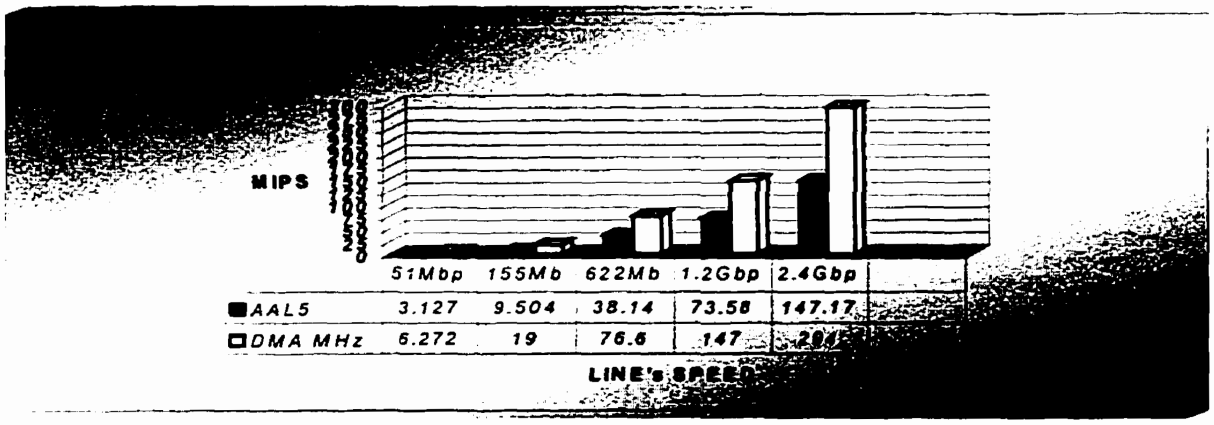
Figure 4.33: ATM Reassembly for AAL3/4 using DMA for data movement (the DMA has the same RISC's clock rate)

Figures 4.34 and 4.35 presents the RISC processing clock rate in order to process the ATM Reassembly for both functions AAL3/4 and AAL5 when the DMA has the double clock rate as the RISC processor.



	51Mbp	155Mb	622Mb	1.2Gbp	2.4Gbp
■ AAL3/4	3.608	11	44.009	84.905	169.81
□ DMA MHz	7.24	21.23	88.05	169.83	[illegible]

Figure 4.34: ATM Reassembly for AAL3/4 using DMA for data movement (the DMA has double RISC's clock rate)



	51Mbp	155Mb	622Mb	1.2Gbp	2.4Gbp
■ AAL5	3.127	9.504	38.14	73.58	147.17
□ DMA MHz	6.272	19	76.6	147	[illegible]

Figure 4.35: ATM Reassembly for AAL5 using DMA for data movement (the DMA has double RISC's clock rate)

In Segmentation function processing, it is clear that the RISC processing speed is become less as we use the DMA with a higher speed (Triple RISC's clock) where all the idle cycle associated with the RISC core processing has eliminated. The VHDL simulation for the Segmentation unit of the network interface has shown that a 68 MHz processor can support 2.4 Gbps lines, when the DMA speed is 213 MHz (triple RISC's clock).

The VHDL simulation for the Reassembly unit of the network interface has shown that an 85MHz processor can support 1.2 Gbps lines supported by the 169 MHz DMA. Clearly, a cost effective RISC core can be used to processes 1.2 Gbps transmission line. Also it is clear that a higher RISC core clock rate could also be used to support higher transmission speed with extra cost. In the next chapter we will see the RISC Architecture that we used in the NI.

Chapter 5

Embedded Pipeline RISC Core for ATM Network Interface

5.1 Introduction

During the VHDL simulation, two RISC cores have been used and supported with DMA in order to process all the AAL3/4 and AAL5 function. The RISC with 85 MHz has been found to be capable of supporting a network interface of 1.2 Gbps and 2.4 Gbps for Reassembly and Segmentation function, respectively. A network interface with high speed can still be supported with the use of the RISC core based NI by using a faster RISC core. In this chapter, we will introduce our three-stage pipeline RISC architecture and describe how it provides the requirements of high-speed ATM network interface.

5.2 Developing RISC core for ATM N I Processing.

The development of a specialized RISC core can generally be done in a short period of time and at lower cost than a general-purpose core. The RISC core, required for ATM interfaces design, is optimized for this application. Hence, some parts, which might be used in RISC core to support the general-purpose applications, may not be required for the ATM network interfaces design. For example, the Floating-Point Unit is not necessary for network interfaces. Also, the use of the data cache is not required since it will not help to improve the performance of the RISC core for this application. The elimination of these units will help to make the core simple to develop at a low cost. In addition, the limited number of instructions that are required to support the ATM interfaces processing can reduce the size of the control unit, improve the speed of such a core, and reduce its complexity.

In order to be able to use the RISC core for different types of network interface, the RISC core should be designed with the Hardware Description Language, VHDL, and that will make such porting operations possible and easy.

5.3 VHDL-Based RISC core

5.3.1 RISC Pipeline

RISC pipelines divide the execution of an instruction into a number of steps, or pipeline stages. The depth of a pipeline corresponds to the number of pipeline stages (Figure 5.1). The schematic capture of the pipeline stages is shown in Appendix A Figure A.7.

The NI RISC core has been designed to execute one instruction in three-pipeline stage:

a) Fetch an instruction from local memory (Fetch stage).

b) Decode/execute the instruction and registers read (Decode/Execute stage).

c) Store results back into the destination register (write back, or W/B, stage).

The RISC fetches instructions which are used to run the ATM protocol program from local memory, Decoding and Executing stage is done to execute the running instruction that has been fetched by the first stage of the pipeline. The last stage of the RISC's pipeline is W/B, in which the data is written to the RISC's register. Some instructions such as Store instruction terminate at Decode/Execute stage.

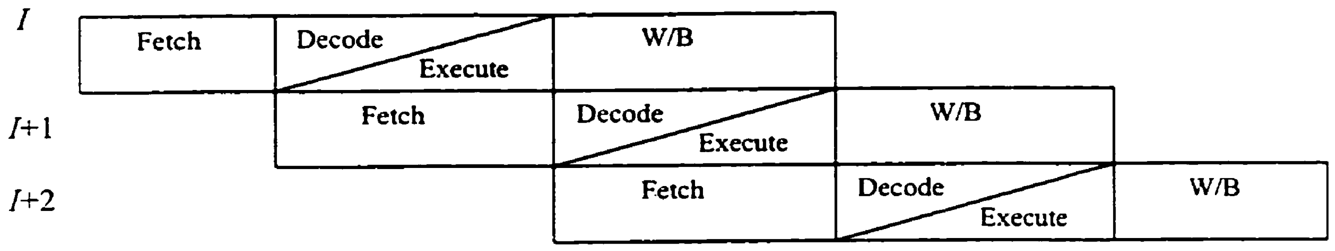


Figure 5.1: Structure of RISC instruction Pipeline

5.3.2 Instruction Representation

During the SPIM simulation, we learn about the suitable instructions that are required for NI. These instructions have been represented in Table 5.1. The instruction format contains the op-code in the first five most significant bits to represent the type of instruction.

Instruction	Functions	op-code	Comments
ADD	r3 <-- r1 + r2	01111	Arithmetic addition
ADDI	r3 <-- r1 + imm	01110	Arithmetic addition immediate
SUB	r3 <-- r1 - r2	00100	Arithmetic subtraction
AND	r3 <-- r1 and r2/imm	00010	Logical AND
LOAD	r1 <--- mem	00001	Load from memory
STORE	r1 --> mem	00101	Store to memory
STCAM	r1-- >(CAM)	00111	Store value at Content Addressable Memory (CAM)
BEQ	r1 = r2/imm ---> label	00110	Branch if equal
BGE	r1 >= r2/imm -> label	01000	Branch greater or equal
BLE	r1 <= r2/imm -> label	00011	Branch less or equal
LCAM	r1 = (CAM) --> r2	01100	Find the match of r1 with CAM contents and store the CAM data in r2

Table 5.1: Type of RISC Instructions.

The description of code instructions group is as follows:

5.3.2.1 Arithmetic and logic operation instructions

They provide computational capabilities for processing numeric data. Logic instructions provide the logical operation such as And, Or, .., etc. The format for arithmetic and logic instruction is as follows:

31 27	26	25 21	20 16	15 11	10 0
Op-code	F	DES	SRR2	SRR1	X

a. Arithmetic/Logic instruction formation (Register-to-Register format)

31 27	26	25 21	20 16	15 0
Op-code	F	DES	SRR1	IMM16

b. Arithmetic/Logic instruction formation (Immediate format)

Where:	SRR1	source register 1
	SRR2	source register 2
	DES	destination register
	F	function bit
	IMM16	immediate value
	X	for future use

Figure 5.2: Arithmetic and logic instruction format

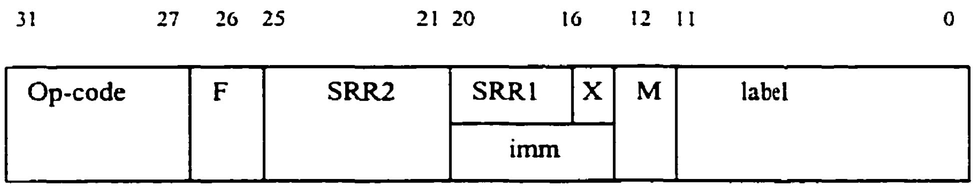
The arithmetic instructions, such as Add, is executed as follows: Add register SRR2 to register SRR1 and store the result into register DES. In the add immediate instruction (addi), the register contents, to which SRR1 refers, will be added to the IMM16 value. The result is stored in register DES. These instructions can be written in the program as:

```
add   r3, r2 , r1     ; Add r2 to r1 and store the result in r3
addi  r4, r3, 10      ; Add r3  to the value of 10 and then store the result in r4
```

As read after write (RAW) data dependency could occur during the program execution, the forwarding mechanism has been implemented to resolve such dependency. For the two preceding instructions, the add instruction stores the result in register r3 where the addi instruction will use r3 as a source operand. During the Decode/Execute stage of addi instruction, the r3 is not updated yet by the add instruction, and thus, an error of calculation will occur. The use of the forwarding mechanism will solve this problem from accruing [DPatt98 and JHen96]. To provide the support that is required by the forwarding mechanism, an F bit is used in the instruction format to initiate the forwarding mechanism if F is '1.' Otherwise no action will be taken. The F bit is set or reset during the program compilation. The compiler can detect whether the forwarding mechanism is required to be initiated or not (more details in section 5.3.3).

5.3.2.2 Branch instructions

Branch instructions are used to test the value of data or the status of a computation before jumping to the label's address. There are three Branch instructions listed in Table 5.1 and all have the same instruction format (Figure 5.3).



Where:		
	label	address in memory (instruction memory)
	imm	immediate value
	F	function bit
	M	immediate/register select
	X	for future use

Figure 5.3: Branch instruction format.

The M bit will be checked by the RISC's controller to distinguish whether the second source operand is an immediate value or a data register. In the case where the M =1, the comparison should be done between SRR1 and the immediate value (value 10). Otherwise, it should be done between SRR1 and SRR2. The F bit is used to control the forwarding mechanism, one of the operand of the current instruction still in the W/B stage of the previous instruction. The forwarding mechanism will be turn on when F is set to 1, to prevent pipeline stalling. The F bit will be reset if no RAW dependency exist between the branch and preceding non-branch instruction.

The branch instruction can be written as follows:

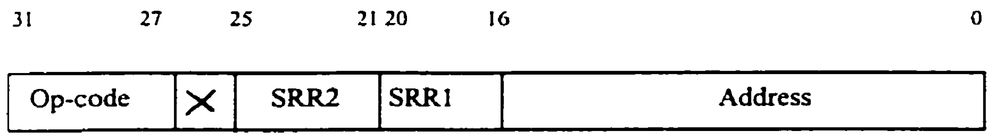
```
beq   r1,r2 , label          ; Branch to label  if contents of r1= contents of r2
beq   r1,10, label           ; Branch to label  if contents of r1= 10
```

The PC will be updated to point to the label's address after checking that the value at r1 is equal to that at r2, or when r1 equal to the immediate value which is 10. The use of the immediate value with the Branch instruction is useful for ATM Payload Type checking (in AAL3/4 the Payload Type is 2-bit, and in AAL5 only 1-bit), or checking if the VCI of the arrived cell is equal value 5.

5.3.2.3 Memory access instructions

These instructions are used to move data between memory and the RISC core registers and these instructions are as follows:

a) Load/Store instruction: To load data from a local memory into a RISC's register, or to store the data register into local memory, the instruction format is as follows:



Where:		
	Address	Memory address
	SRR2	Holds the data that needed to store in local memory (if the instruction is load), in store instruction used as the destination register (to store the data memory)
	SRR1	memory address
	X	for future use

Figure 5.4: Load/Store instruction format.

For Load instruction, the value at address field and the contents of the SRR1 are used to address the source address. The SRR2 is used as the destination register. The same is applied for Store instruction where the memory address is calculated as in the load instruction, while the SRR2 is used as the source register instead of the destination register. The Laod/Store instruction can be written as follows:

```
lw  r1, address (r2)   ; r1 = Memory[r2 + address]

sw  r4, address (r2)   ; Memory[r2+address] = r4
```

b) Load CAM (LCAM): This instruction is used to load data from the CAM after a match for a certain data with the contents of the CAM is found. The instruction format of LCAM is shown in Figure 5.5

31 27	26	25 21	20 16	0
Op-code	S	DES	SRR1	X

Where :

SRR1	holds the data match
DES	store the loaded data from CAM
S	bit signal to the RISC's controller
X	for future use

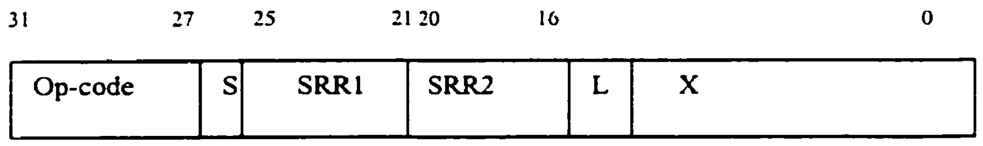
Figure 5.5: LCAM instruction format.

The contents of the SRR1 is used to be matched with the CAM contents, if the match found, the CAM then returns the contents of that matched location. SRR1 contains either VC or VCI-MID which needed to be match with another active identifier that stored in CAM content, if matched exist, then the Start or end Address will send out from CAM. The RISC's controller after reading the S bit, it sends a signal to the CAM 's controller (Start_end_signal) to either send the Start-address or End-address as data out from the CAM to the RISC. DES is a distention register which used to store the CAM output.

If the match is not found, the CAM sends a signal to the RISC (match = 0), the RISC, considers this cell as a lost cell and discardes it. The LCAM instruction can be written as:

```
lcam   r2,r1      ; match what in r1, which could be VC or VCI-MID, with the contents of CAM.
                  ; then  reads CAM data output into r2.
```

c) Store CAM (STCAM): This instruction is used to store data in CAM. The instruction format is shown in figure 5.6.



Where :		
	SRR1	holds the data, such as VC or VCI-MID
	SRR2	holds the stored data (Start or End address)
	S	bit signal to the RISC's controller
	L	type of data that needed to store it in CAM
	X	for future use

Figure 5.6: RISC instruction-set format for CAM operation (stcam).

The register SRR1 could hold the value for VC or VCI-MID that is required to be stored as new entry in the CAM. After a match is found, the instruction can be used to store the Start or End-address. The SRR2 contains the data that the RISC needs to store in CAM, i.e. the Start-address or the End-address. The RISC's controller will send a signal to the CAM's controller to store the data as a Start or End address, or to store the new entry (see section 4.2.2). If the S = 1 and L = 0 then store a new entry in CAM, in the hand if the s = 0 and L = 1 this means store Start-address, if it is 00 then store End address. An example of the SCAM instruction as follows:

```
stcam r9, r3        ;Store the contents of r9 in CAM where r3 has the value of CAM entry to be
                    ; matched such as VC or VCI-MID.
```

5.3.3 Pipeline Hazard

In the instruction stream, hazard is the prevention of the next instruction from being executed during its designated clock cycle. Clearly, hazard reduces the RISC performance. Hazard types include:

1- Control hazards.

2- Data hazards.

The Control hazard could occur during the condition branch instruction execution. The decision about whether the branch is taken or not taken does not occur until the result of the comparison is completed (in Decode/Execute pipeline stage). The fetched instruction after the condition branch instruction will be flashed from the pipeline if the branch is taken. Such operation is called branch penalty. Clearly, the branch penalty will reduce the pipeline performance. The branch-delay technique is used to reduce such problems and is as follows:

```
i       and  r1 , r7 , 0x00000000
i+1     add r6, r2, r3
i+2     beq r6,r4, label
```

a: Before scheduling the branch-delay slot

```
i       add r6, r2, r3
i+1     beq r6,r4, label
i +2    and  r1 , r7 , 0x00000000
```

b: After scheduling the branch-delay slot

Figure 5.7: Scheduling the branch-delay

In Figure 5.8 (a) shows the code before scheduling where *add* instruction will execute before the branch instruction. The *add* instruction in Figure 5.8 (a) is considered as independent instruction. In Figure 5.8 (b) the *add* instruction is used as a delay slot which is scheduled to be executed after the branch instruction. In this case, the *and* instruction will be executed in either way (if the branch is taken or not) and that will not affect the pipeline's performance.

The frequency of occurrence for conditional branches for Reassembly unit for both AAL3/4 and AAL5 is shown in Table 5.4. The branch-delay slot technique can be used to eliminate the effect of performance degradation due to conditional branch instructions. We found that a useful instruction can be managed for every conditional branch instruction for the processing of both AAL3/4 and AAL5.

Message type	AAL3/4	AAL5
SSM	8	8
BOM	9	8
COM	9	8
EOM	7	8

Table 5.2: Occurrence of the conditional branch for AAL3/4 and AAL5 Reassembly Processing

The data hazards could occur when an instruction being executed in the current pipeline stage requires a result that still unavailable of an instruction executed in an earlier pipeline stage. The following portion of the ATM protocol is an example where there is a data hazard:

```
.
.
.
i     lw r4, 0(r1)                 ; load a free pointer from the CB
i+1   and r1, r6, 0x000ffff0       ; Mask VCI from ATM header (r6) and store the result in r1
i+2   ble r1, 5, signaling         ; Check if the current cell is signaling,
i+3   and  r3,r6,0x0000000E        ; Mask the PT from ATM header (r6) and store the result in r3
i+4   beq r3, 0x00000002, EOM      ; Check if the current cell is the EOM/SSM  cell of
                                   ; CPCS-PDU by checking the R3's value
i+5    and r5, r6, 0x0ffffff0      ; Mask The VC (VCI,VPI) from ATM header (r6) and store the
                                   ; result in R5
.
.
.
```

The *and* instruction, at *i+1*, writes the value of r1 in the W/B pipeline stage, where the *ble* instruction at *i+2* reads the value during its Decode/Execute stage. Clearly, the *ble* instruction will have the data hazard problem. Unless precautions are taken to prevent it, the *ble* instruction *i+2* will read the wrong value of the r1. Data hazard also happen between *i+3* and *i+4* instruction, when *i+4* is trying to use r3 for the comparison.

Clearly, this data hazard happened quite often in the NI codes. A support should be provided for NI design in order to reduce its impact. Otherwise, a severe reduction in performance will occur.

There are different techniques that can be used to reduce or even eliminate the performance reduction due to the data hazard (or dependency). One technique is forwarding or injecting. By using the injecting technique a useful instruction will be placed before instruction that required an information where the needed information still not provided yet by the proceeding instruction. In our work, we decided to use the injection of a useful instruction to avoid the hazard. The same code in the previous example could be written as follows:

```
.
.
.
i   (i+1 previously) and r1, r6, 0x000ffff0   ; Mask VCI from ATM header (r6) and store the result in
                                               ; r1
i+1 (i previously)   lw r4, 0(r1)              ; load a free pointer from the CB
i+2                  ble r1, 5, signaling      ; Check if the current cell is signaling,
i+3                  and  r3,r6,0x0000000E     ; Mask the PT from ATM header (r6) and store the
                                               ; result in R3.
i+4 (i+5 previously) and r5, r6, 0x0ffffff0    ; Mask The VC (VCI,VPI) from ATM header (r6)
                                               ; and store the result in r5.
i+5 (i+4 previously) beq r3, 0x00000002, EOM   ; Check if the current cell is the EOM/SSM  cell
                                                 ; of CPCS-PDU by checking the r3's value
.
.
.
```

By using the injecting technique to avoid the data dependency problem, the pipeline then will not forced to stall during the NI code execution. The frequency of data hazard occurrence for both AAL3/4 and AAL5 Reassembly is shown in Table 5.2 and 5.3. Although the occurrence appears to be minor, the impact or reduction of performance is significant. After rescheduling the program that has been written for ATM Reassembly, we still have one RAW hazard for AAL5 within BOM and COM out of the total cycle required for the BOM (where the total instruction needed is 26 cycles). One RAW within SSM for AAL3/4 is also required for the BOM (where the total instruction needed is 30 cycles).

AAL5 message type	**RAW Hazard**
BOM and COM	1
SSM and EOM	0

Table 5.3: Occurrence of the Read After Write (R/W) hazard for AAL5 Reassembly Processing

AAL3/4 message type	**RAW Hazard**
BOM, COM, EOM	0
SSM	1

Table 5.4: Occurrence of the Read After Write (R/W) hazard for AAL3/4 Reassembly Processing

Since most independent instructions are used to avoid the control hazard, we unable to find a useful instruction to be injected after these instructions cause the hazard. Still, adding the forward mechanism in our simulator is important to eliminate the data hazard that may occur [DPatt98 and JHen96]. An example of the forward mechanism used in our simulator is as following:

```
I       sub  r3, r1, r2           ; Add r1  to r2 and store result in r3
             |
             | Forward R3
             v
I+1     beg r3, 5, send signal      ; If  the r3 is equal to value  5 then jump to the send
                                    ; signal locations
```

Latch the result of the ALU (rl - r2), and then send the latched data at register (r3) to the ALU to compare it with value 5 (during the Decode/Execute of the current instruction). The schematic capture for the forwarding mechanism is shown in Appendix A, Figure A.8. The forwarding hardware receives a signal from the RISC's controller indicating that the current instruction needs the latched data of the previous instruction. The latched data is sent to the Decode/Execute stage of the current instruction (Figure 5.9).

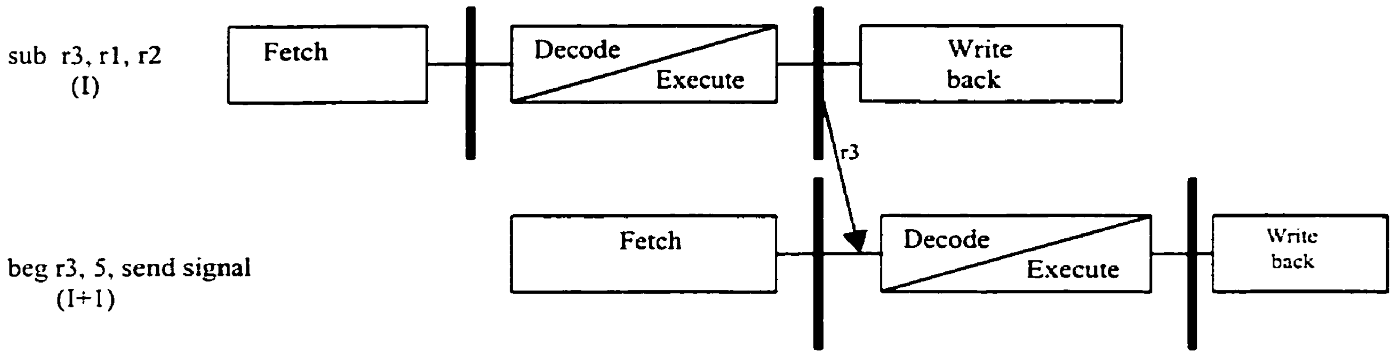


Figure 5.8: Minimize Data Hazard by latching the output of the ALU to be read within next instruction (forward mechanism).

5.4 RISC's Registers

In our implementation, the RISC's instruction format has three register operands. We will need to read two data words from the register file and write one data word into the register file for each instruction. For each data word to be read from the registers, we need an input to the register file that specifies the register number to be read and an output from the register file that will carry the value that has been read from the registers. To write a data word, we need two inputs. The first input needed specifies the register number to be written. The second input supplies the data to be written into the register.

Thus, we need a total of four inputs (three for register number and one for data) and two outputs (both for data) (Figure 5.10). The register files VHDL based is shown in Appendix A Figure A.9. The register file always outputs the contents of whatever register numbers are on the Read register inputs. Read registers is controlled by a specific signal, called *R(R1,R2)*, which must be asserted a read command to the specified register. The write register is controlled by a specific write control signal, called *W(for W/B register)*, which must be asserted by RISC's controller to write data into certain register.

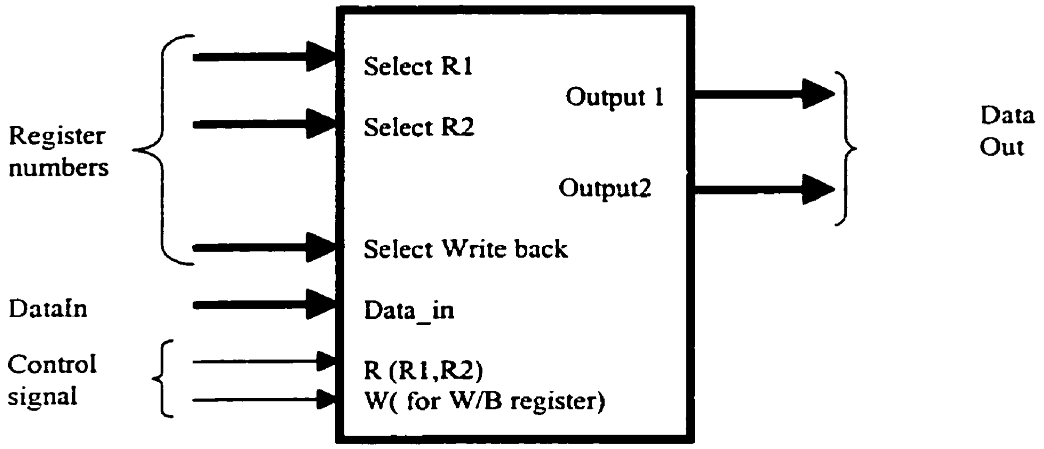


Figure 5.9: RISC register file

The register number inputs are 5 bits wide to specify 1 of existing registers, whereas the data input and two data outputs are each 32 bit wide. The size of the register files differ between the Reassembly and Segmentation functions. The sizes of register files for the Reassembly unit and for the Segmentation unit are shown in Table 5.5. Clearly, the register file within RISC processor in the Reassembly unit is larger than the Segmentation unit because the Reassembly function has extra registers just needed to hold the specific information (during the setup operation). This information includes two registers to hold the CB's pointers (the head and tail of CB). The other registers to hold the address of the FIFOs (FIFO1, 2, 3 and 4).

Functions unit	**Register size**
Reassembly	28 register
Segmentation	20 register

Table 5.5. Register file size for Segmentation and Reassembly units

5.5 The Component Needed With RISC cores.

Both RISC cores may need other components helping with SAR processing. Table 5.6 shows the component needed for each side.

Processing core Performance	Segmentation	Reassembly
DMA	Required	Required
CAM	Required	Required for the Active connection
Local bus	32-bit word bus	32-bit word bus
Circulation Buffer	Not required	Required to Store the free pointer spaces
Number of FIFO(s)	1	4
SRAM Cell buffer	Required	Required

Table 5.6 : Shows the component needed for segmentation and Reassembly functions

Chapter 6

Conclusion and Future work

We have presented computer simulation to measure the amount of processing required by the ATM network interface for both AAL 3/4 and AAL 5. The VHDL simulator has shown that the processing requirements for the data movement of the Segmentation and Reassembly units can be reduced by using DMA controller. Such controller must runs at two to three times the speed of the embedded RISC could eliminate all RISC's idle cycles.

Also, the simulation results have shown that a cost effective embedded RISC core can efficiently provide network interface with the processing that required to support a wide range of transmission line speed. A 70MHz RISC core can support the segmentation unit processing for up to 2.4Gbps transmission speed, while a core running at 85MHz is found to be suitable for the Reassembly unit processing for up to 1.2Gbps line speed. These results are based on the use of a specialized RISC core that we developed and simulated for ATM NI applications. Such core has three stages pipeline supported with forwarding mechanism, instruction set of only 11 instructions, a register file of 20 register for Segmentation and 28 for Reassembly.

As the future work for this thesis, we would like to investigate the support for AAL1 and AAL2 by our NI model. Such support will require the protocol functions modification and not the NI architecture. Also, the ATM NI can be extended to process the upper layer of the ATM protocol the Convergence Sub-layer (CS) layer. Such support can be useful to have direct network-to-device communication with minimum interfering from the host processor.

The use of the RISC processing core, for other type of network interface can also be investigated in the future.

Bibliography

[AElka99| A. Elkateeb and M. Elbeshti, "An Evaluation of the ATM Protocols Processing Requirements for Network Interfaces Design", proceeding of the 1999 symposium on Performance Evaluation of Computer and Telecommunication Systems, Chicago, July 1999. PP. 13- 16.

[AElka00] A. Elkateeb and M. Elbeshti, " A Study of the Use of the RISC-Core for ATM Network Interfaces", Computer Communications Journal, Vol. 23 No. 2, February/March 2000.

[DBru93] Bruce S Davie " The Architecture and Implementation of a High-Speed Host Interface" IEEE Journal on Selected Area in Communication, Vol. 11 No. 2, February 1993

|CKim98] C. Kim and et al. " Design and Implementation of A High-Speed ATM Host Interface Controller", *Proceeding* of ICOIN, 1998 PP 525-528

[Comp98] Computer Architecture News " A QoS Communication Architecture for Workstation Clusters" Vol. 26, No. 3 - June 1998

[ZDit97] Dittia Zubin D, Parulkar Guru M and Cox Jerome R "The APIC Approach to High Performance Network Interface Design: Protected DMA and Other Techniques," *To appear in the Proceedings of IEEE Infocom 1997*

[DPatt98] D. Patterson and J. Hennessy, *Computer Organization and Design the Hardware/Software interface,* Morgan Kaufmann Publisher Inc., 1998.

[ECoop91] Eric Cooper, Onat Menzilcioglu, Rebert Sansom and Francois Bitz " Host Interface Design for ATM LANs" IEEE 1991

[CGeor97] C. Georgiou and C. Li, "Scalable Protocol Engine for High-Bandwidth Communications " *Proceedings of the 1997 IEEE international Conferences on communications*, Montreal June 1997. PP1121-1126

[GPart94] Graig Partridage *"Gigabit Network"* 1994 by Addison-Wesley Publishing company

[ITUR93] ITUT-T Recommendation I.361 was revised by the ITU-T Study Group XVIII (1988-1993) and was approved by WTSC (Helsinki, March 1-12, 1993).

[ITUT95] ITU-T Study Group XVIII, "Recommendation Q.2931 - Broadband ISDN - B-ISDN application protocols for access signaling", ITU, 1995

[ITUT93] ITU-T Study Group XVIII, "Recommendation I.363 - B-ISDN ATM Adaptation Layer (AAL) Specification", ITU, March 1-12, 1993

[ITU-93] ITUT-T Recommendation I.432 was revised by the ITU-T Study Group XVIII (1988-1993) and was approved by WTSC (Helsinki, March 1-12, 1993).

[JHen96] J. Hennessy and D. Patterson and, *Computer Architecture a Quantitative Approach* Morgan Kaufmann Publisher Inc., 1996

[LPete96] Larry Peterson and Bruce S. Davie " *Computer network* " A system Approach 1996

[PMart95] Martin de Prycker " *Asynchronous Transfer Mode Solution for Broadband ISDN*" Printed and bound in Great Britain 1995

[RHosb99] Richard F. Hosbson, P.S. Wong " A Parallel Embedded-Processor Architecture for ATM Reassembly" IEEE/ACM *Trans. On Networking,* Vol. 7, No. 1 February 1999

[STraw93] S. Traw and Jonathan Smith " Hardware/Software Organization of a high performance ATM Host interface" IEEEJSAC Vol.11 , No. 2 Feb 1993.

[KSka96] Keven Skahill *"VHDL for Programmable Logic"* 1996 by Addision-Wesley Publishing

[Dperr98] Douglas Perry *" VHDL third edition"* 1998 published by McGraw-Hill Companies.

[Xilin99] Xilinx home page (http://www.xilinx.com)

[Xilin98] Xilinx foundation series software ver 1.5 1998/1999

Appendix A

VHDL Simulation Diagrams

In this Appendix, the schematic diagrams for every unit of the network interface have been presented.

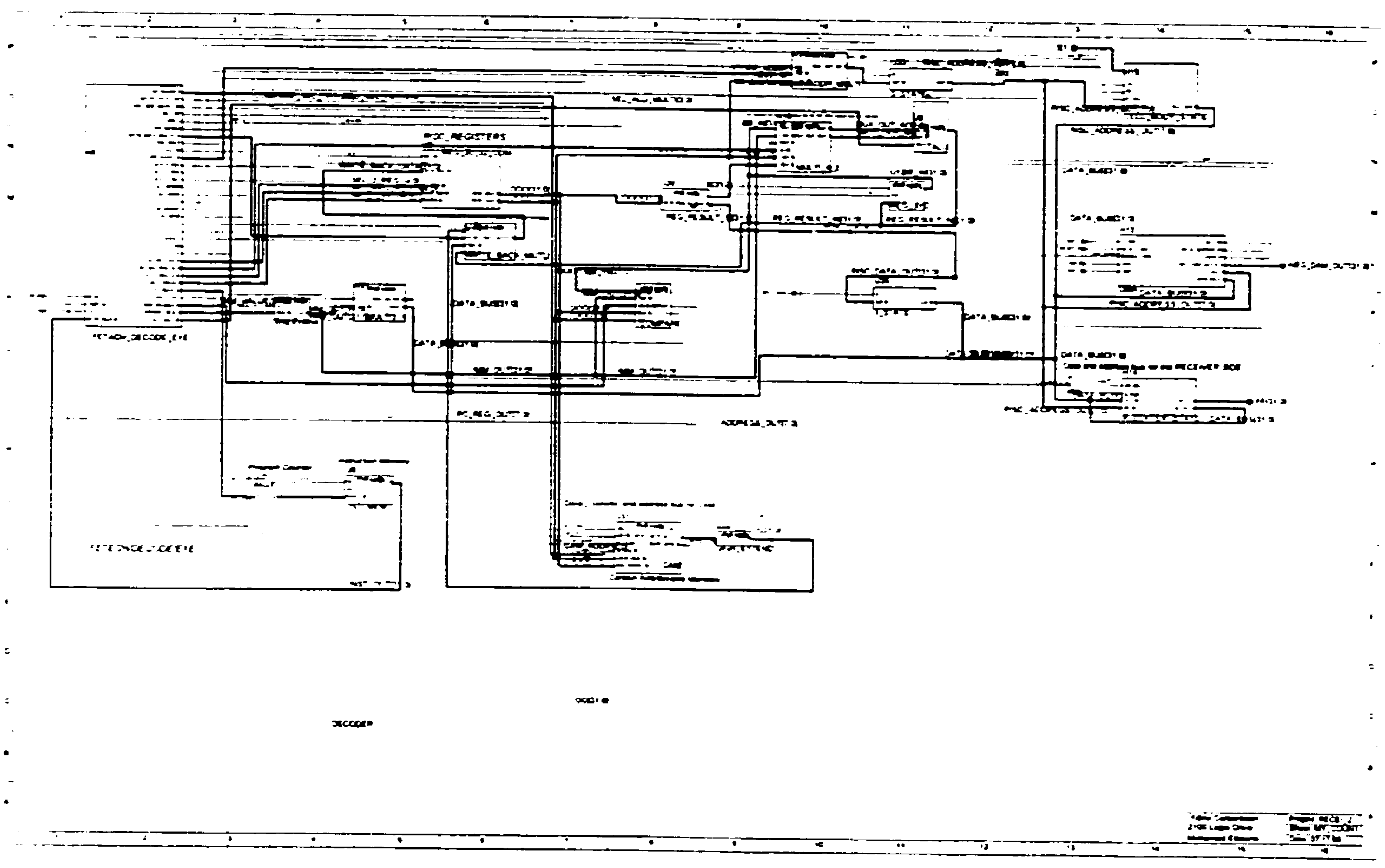


Figure A.1 VHDL based ATM Network Interface architecture

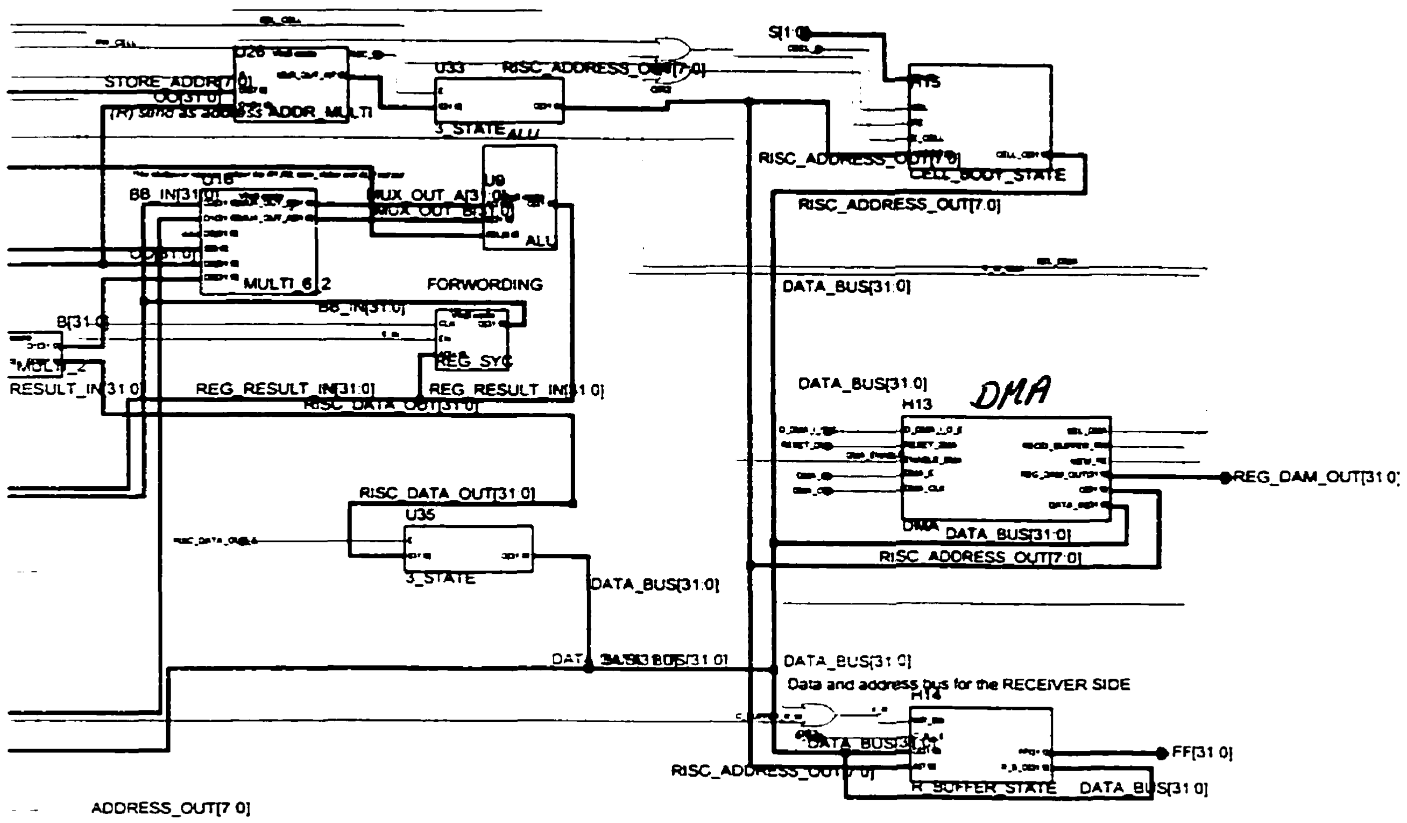


Figure A.2: Location of the DMA controller in the NI architecture.

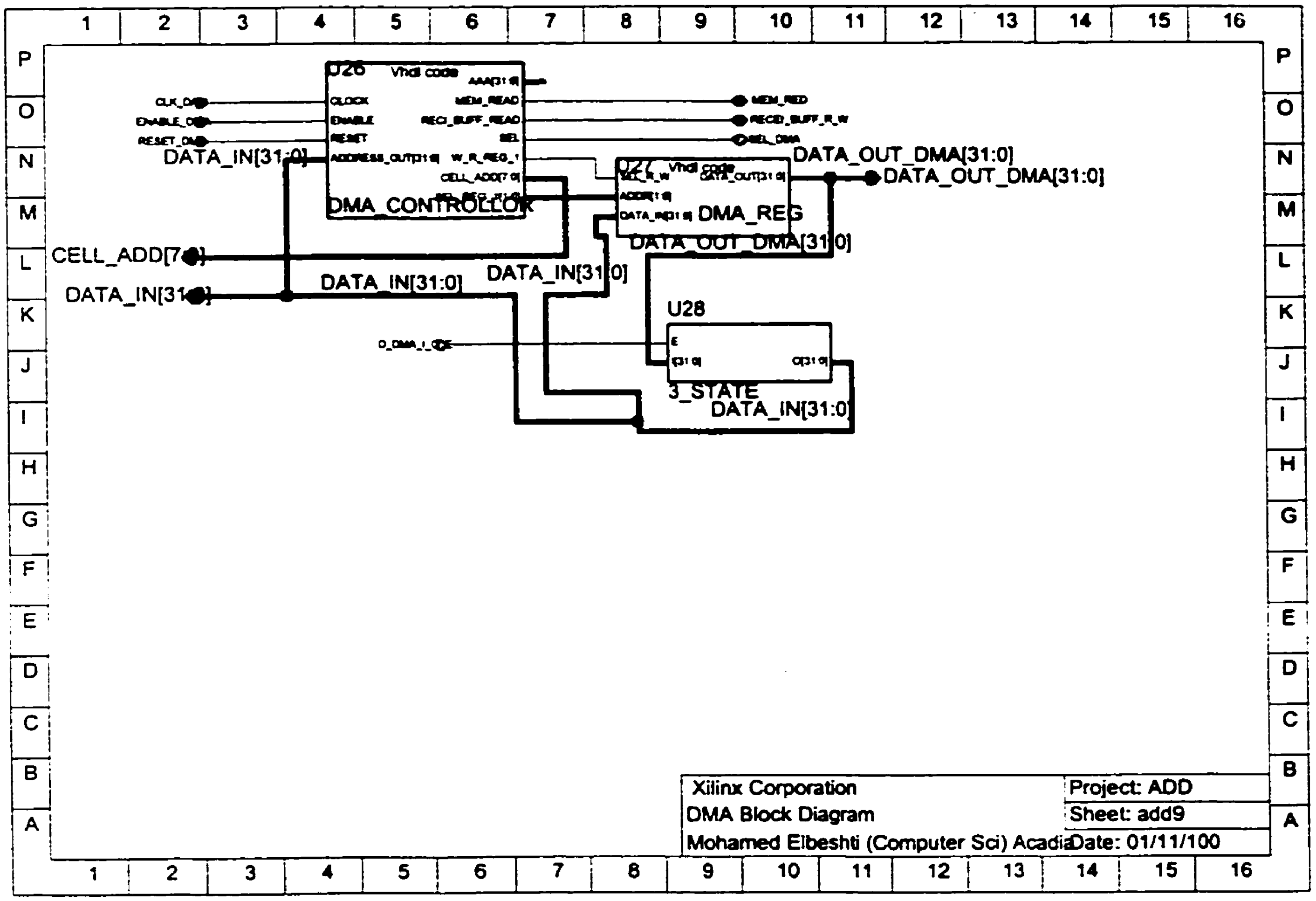


Figure A.3: DMA schematic diagram

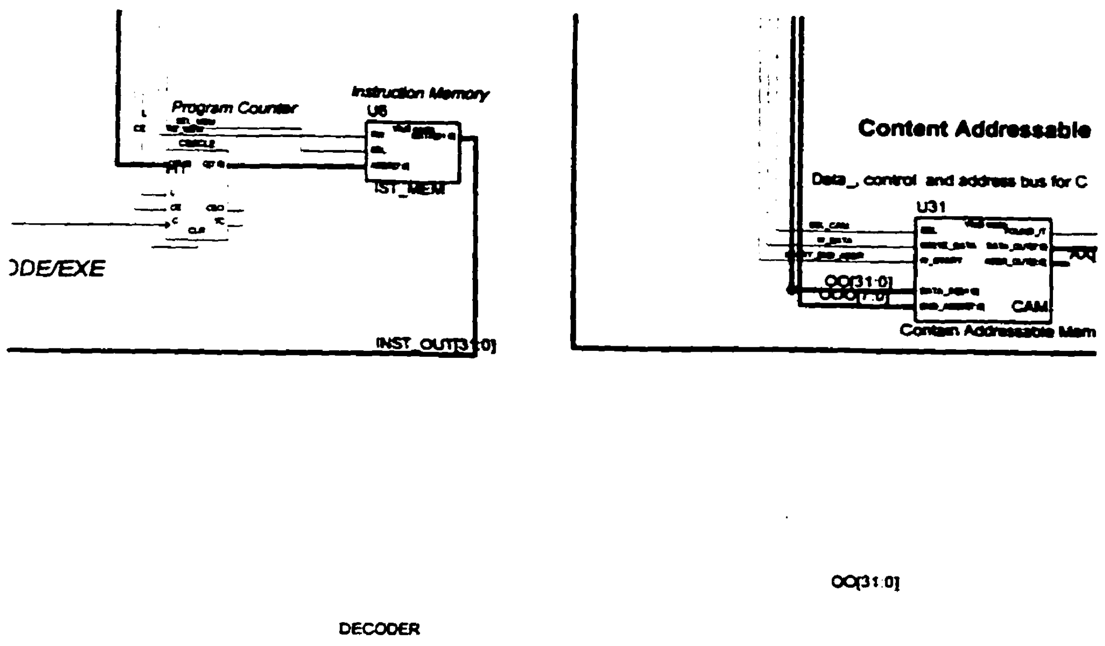


Figure A.4: CAM structure

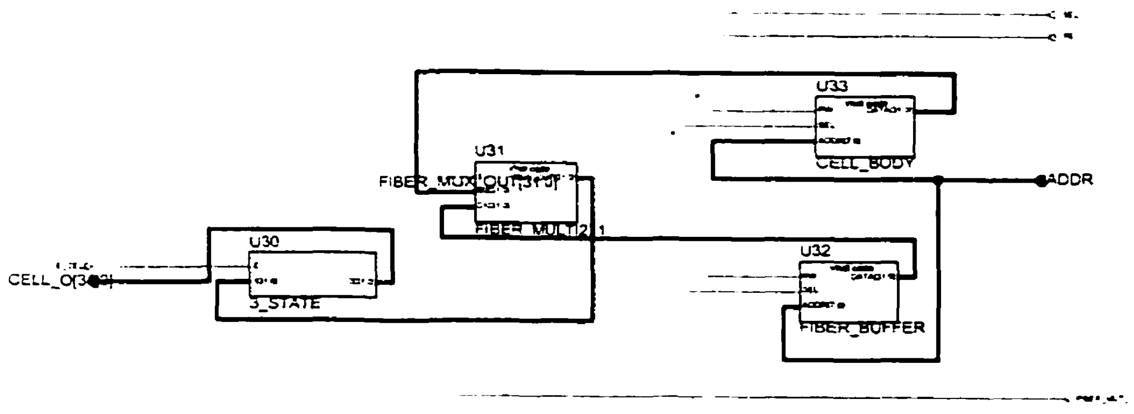


Figure A.5: Receiver Buffer Interface (RBI)

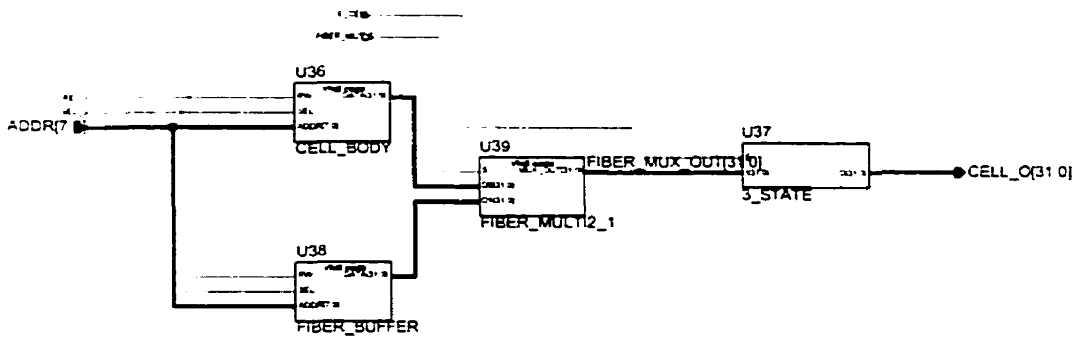


Figure A.6: Sending Buffer Interface (SBI)

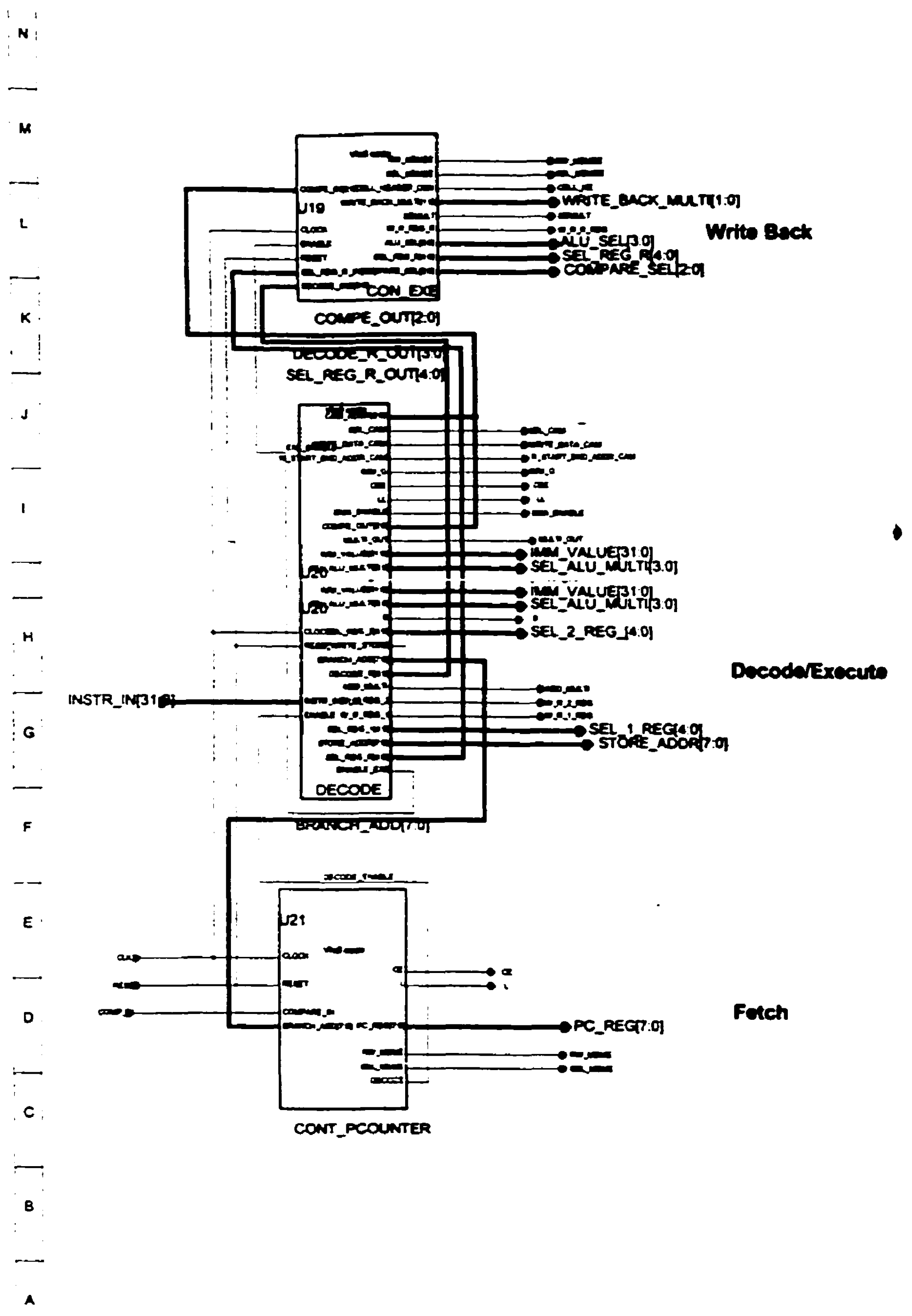


Figure A.7: Structure of RISC instruction VHDL based pipeline

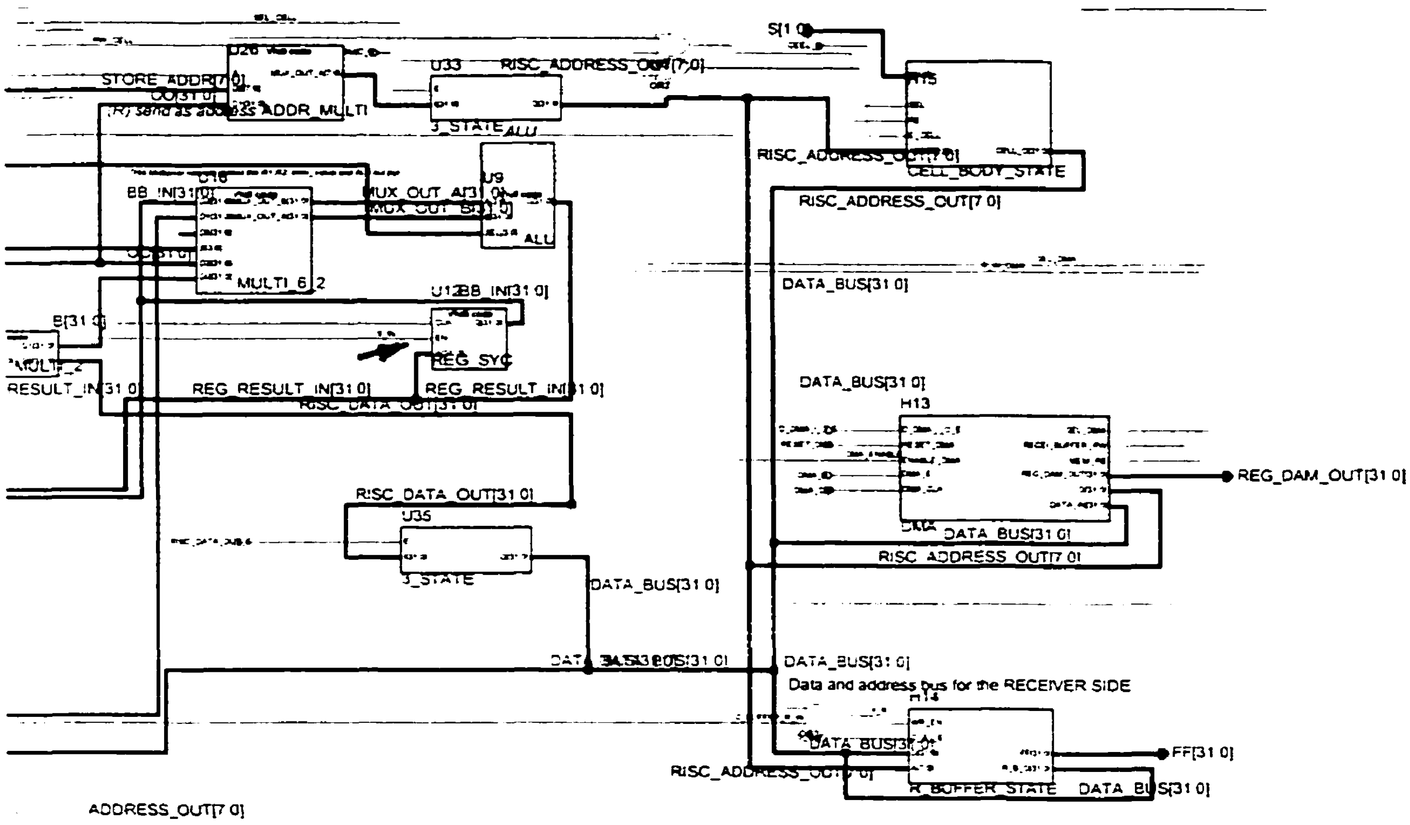


Figure A.8: Minimize Data Hazard by latching the output of the ALU by forwarding hardware (U12) to be read within next instruction (forward mechanism).

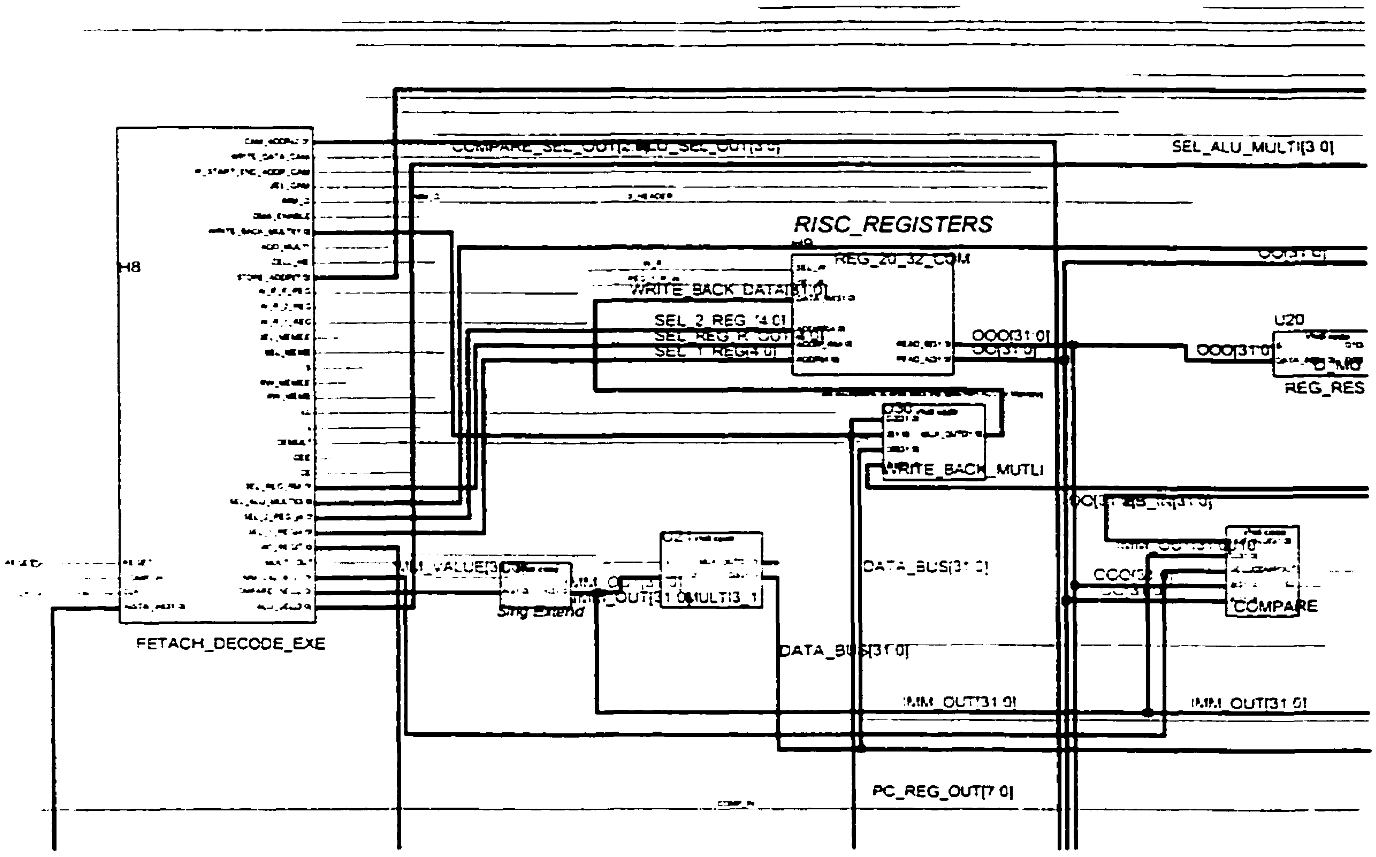


Figure A.9: RISC register file

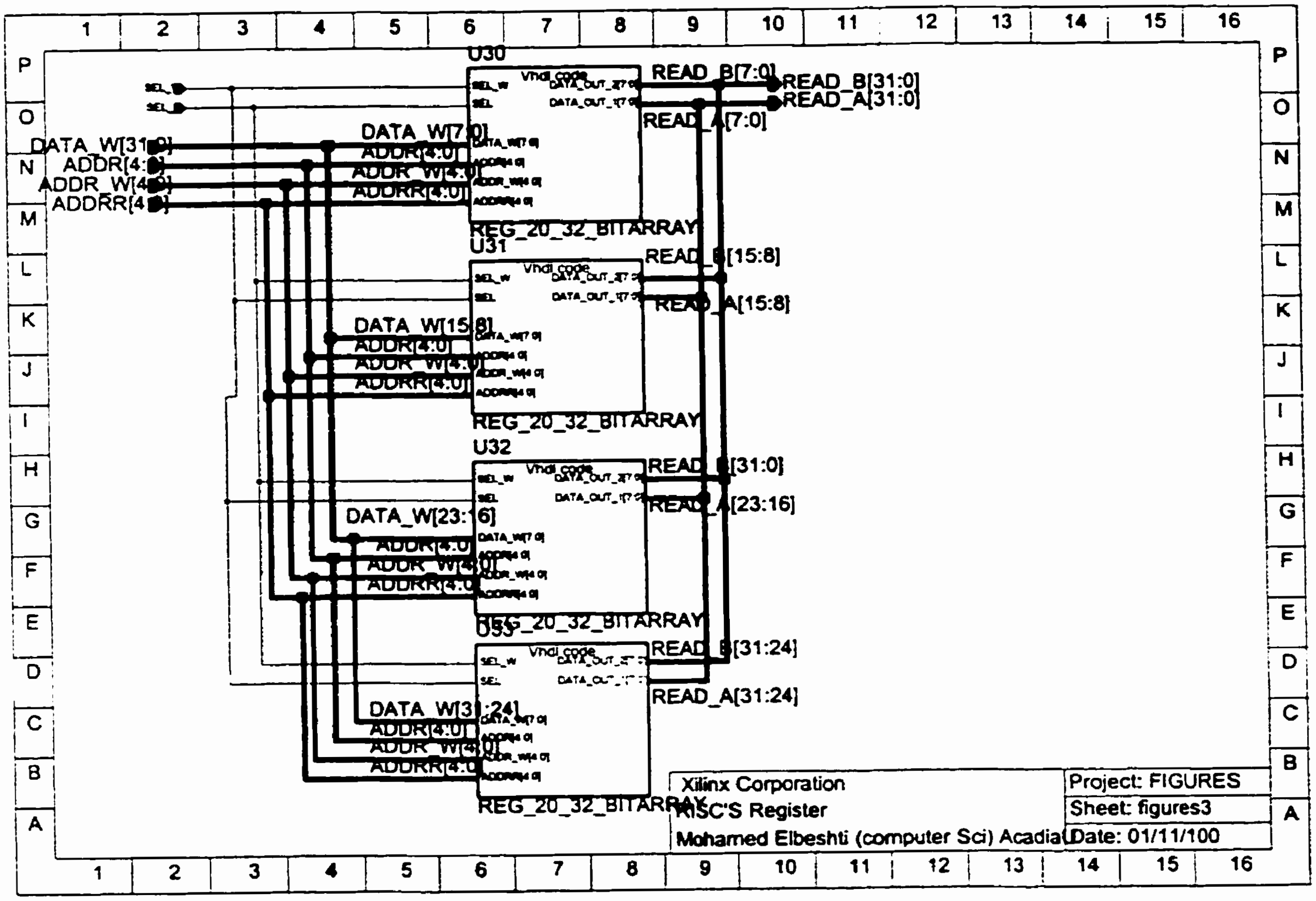


Figure A.10: RISC register file structure format