Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-2010-01

M.Sc. Thesis

Specification and Implementation of a DMA Controller In an Embedded System

Daoxin Li B.Sc.

Abstract

The fast growing of In-Car entertainment application leads to an increasing challenge for both data computation and data communication, which are managed by the microprocessor. The thesis project is the third stage of a continuous Direct Memory Access(DMA) Controller project in NXP Semiconductors for the purpose of specifying and implementing a DMA Controller to take over data communication tasks from the microprocessor. In the first step of the thesis, a test principle was investigated to fully test the existing results, but the simulation results of the Core Unit did not satisfy the requirements. The Core Unit of the DMA Controller is responsible for the sequential-single transfer and burst transfer involving wait states. The existing specification and implementation were analyzed, and a number of possible approaches for improvements were identified. During the second step, the Core Unit was re-specified according to these approaches, and fully implemented using VHDL to fulfill the requirements. After the Core Unit design, the functions of Linked List transfer was specified with Hatley and Pirbhai methodology. The Linked List Unit, which manages the Linked List transfer, was specified to support both the Static and Dynamic Linked List transfer. This specification provides an essential base for the future implementation. The implementation of the Core Unit was tested with Simvision following the proposed test principle. The results satisfied the function requirements. Thus, the specification was proved to be feasible. Additionally, the Core Unit was synthesized using Cadence Ambit. The number of the equivalent gates of a Core Unit Cell is 3k, which is smaller than the currently used DMA Controller in NXP Semiconductors.

Specification and Implementation of a DMA Controller In an Embedded System

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

MICROELECTRONICS

by

Daoxin Li B.Sc.
born in Yongchuan, China

This work was performed in:

Circuits and Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Specification and Implementation of a DMA Controller In an Embedded System"** by **Daoxin Li B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: Feb. 16th, 2010

Chairman: _____
Prof.Dr.Ir. Alle-Jan van der Veen

Advisor: _____
Dr.Ir. Rene van Leuken

Committee Members: _____
Dr. Sorin Cotofana

_____
Harpreet Singh Bhullar

Abstract

The fast growing of In-Car entertainment application leads to an increasing challenge for both data computation and data communication, which are managed by the microprocessor. The thesis project is the third stage of a continuous Direct Memory Access(DMA) Controller project in NXP Semiconductors for the purpose of specifying and implementing a DMA Controller to take over data communication tasks from the microprocessor.

In the first step of the thesis, a test principle was investigated to fully test the existing results, but the simulation results of the Core Unit did not satisfy the requirements. The Core Unit of the DMA Controller is responsible for the sequential-single transfer and burst transfer involving wait states. The existing specification and implementation were analyzed, and a number of possible approaches for improvements were identified. During the second step, the Core Unit was re-specified according to these approaches, and fully implemented using VHDL to fulfill the requirements. After the Core Unit design, the functions of Linked List transfer was specified with Hatley and Pirbhai methodology. The Linked List Unit, which manages the Linked List transfer, was specified to support both the Static and Dynamic Linked List transfer. This specification provides an essential base for the future implementation.

The implementation of the Core Unit was tested with Simvision following the proposed test principle. The results satisfied the function requirements. Thus, the specification was proved to be feasible. Additionally, the Core Unit was synthesized using Cadence Ambit. The number of the equivalent gates of a Core Unit Cell is 3k, which is smaller than the currently used DMA Controller in NXP Semiconductors.

Acknowledgments

First, I would like to thank my advisor, Dr.Ir. Rene van Leuken in TU Delft Circuit and System group and Harpreet Singh Bhullar in NXP Semiconductors for their guidance. Mr. van Leuken created my internship opportunity in NXP Semiconductors and supported me a lot during the internship period. Mr. Bhullar continued to guide me after my first advisor in NXP Semiconductors was laid off. He helped me to make a suitable schedule for the last two months of my internship, and also gave me lots of good advice in many fields.

Second, I am also grateful to Jos Teunissen and Rene van den Berg. Mr. Teunissen was my first advisor in NXP Semiconductors. He led me into a new design methodology and also shared his experiences to help me to fit into Dutch environments better. Mr. van den Berg was my co-advisor together with Mr. Bhullar. His insight into the project made me to understand my project easier.

Third, I would also like to thank Prof. Dr.Ir. Alle-Jan van der Veen. His attitude towards his work inspirited me to continue going forward. He was always patient when I came to him for advice.

Fourth, I want to express my appreciation to the engineers in NXP Semiconductors. I really learned a lot from them. It's a great experience to work with them.

Finally, during the whole periods of thesis project, my friends and my family were always there when I needed them, and supported me without selflessly. It's my pleasure to have all of you in my life. Thank you all.

Daoxin Li B.Sc.
Delft, The Netherlands
Feb. 16th, 2010

Contents

Abstract **v**

Acknowledgments **vii**

1 Introduction **1**
1.1 Project Motivation . . . 1
1.2 Project Goals . . . 2
1.3 Results and Contribution . . . 2
1.4 Thesis Organization . . . 3

2 Background **5**
2.1 Data Communication of the Design ICs . . . 5
2.1.1 Data Communication and Computation of the Design ICs . . . 6
2.1.2 Methods of data Streaming computation module . . . 8
2.1.3 Comparisons of the Three Methods . . . 11
2.2 Introduction of DMA Controller . . . 11
2.2.1 Architecture of A General DMA Controller . . . 11
2.2.2 Function Descriptions of A DMA Controller . . . 14
2.2.3 Basic Operation Modes and Transfer types . . . 15
2.2.4 Applications and Trends of DMA Controller . . . 17
2.3 Limitations of Currently used DMA Controllers in NXP . . . 17
2.3.1 DMA Controller 1 used in NXP . . . 17
2.3.2 DMA Controller 2 used in NXP . . . 18
2.3.3 Problems of these DMA Controllers . . . 19
2.4 Conclusions . . . 20

3 Problem Analysis of Existing Results **21**
3.1 Introduction of Existing Results . . . 21
3.2 Problem Analysis of Existing Results . . . 23
3.2.1 Test Principle Setup . . . 23
3.2.2 Simulation Results . . . 25
3.2.3 Problem Analysis . . . 25
3.2.4 Possible Approaches for Improvements . . . 26
3.3 Conclusions . . . 28

4 Core Unit Design Implementation **29**
4.1 Core Unit Conceptual Architecture Specification . . . 29
4.1.1 Core Unit Context Diagram . . . 29
4.1.2 Core Unit Requirements and Constrains . . . 30
4.2 Logical Architecture Specification and Design . . . 31
4.2.1 Process Model Hierarchy . . . 31
4.2.2 Top Level Logical Architecture Specification . . . 31

4.2.3 Core Unit 1 Specification and Design 32
4.2.4 Core Unit 2 Specification and Design 34
4.3 Conclusion 43

5 Linked List Design Implementation **45**
5.1 Conceptual Architecture Specification and Design 45
5.1.1 Introduction of Linked List Transfer 45
5.1.2 Conceptual Architecture Specification and Design 47
5.2 Logical Architecture Specification and Design 49
5.2.1 Process Model Hierarchy 49
5.2.2 Logical Architecture Specification 49
5.3 Conclusions 59

6 Simulation and Synthesis **61**
6.1 Simulation and Results 61
6.1.1 Simulation Environment Setting 61
6.1.2 Simulation Results 62
6.2 Synthesis Result 63
6.3 Conclusions 64

7 Conclusions and Future Work **65**
7.1 Conclusions 65
7.2 Future Work 66

A Appendix **67**

B Appendix **69**

C Appendix **71**

Bibliography **72**

List of Figures

2.1 In-Car Entertainment System 5
2.2 Simplified In Car Entertainment System 6
2.3 Single Entertainment Format 6
2.4 Multiple Entertainment Format 7
2.5 Multiple Format Separate Module 7
2.6 Polling based data transfer 8
2.7 Interrupt based data transfer 9
2.8 DMA based data transfer 10
2.9 Simple Structure of a DMA Controller 13
2.10 DMA Controller 1 18
2.11 Simple DMA Controller 18
2.12 DMA Controller 2 19

3.1 First Stage DMA Controller Context Diagram [1] 21
3.2 Second Stage Top Level Data Flow Diagram [2] 22
3.3 Location Examples of Wait States Insertion 24
3.4 Example of Incorrect Burst Transfer 26

4.1 Core Unit Context Diagram 30
4.2 Core Unit Process Hierarchy 31
4.3 Core Unit 0 Data Flow Diagram 32
4.4 Core Unit 0 Control Flow Diagram 32
4.5 Core Unit 1 Analysis 33
4.6 Core Unit 2 Data Flow Diagram 35
4.7 Core Unit 2 Control Flow Diagram 36
4.8 Core Unit 2 FSM Idle State 37
4.9 Core Unit 2 FSM Sequential-Single Read State 38
4.10 Core Unit 2 FSM Sequential-Single Write State 39
4.11 Core Unit 2 FSM Burst Read State 41
4.12 Core Unit 2 FSM Burst Write State 42

5.1 Static Linked List Process [1] 46
5.2 Dynamic Linked List Process 46
5.3 Linked List Context DFD 48
5.4 Linked List Context CFD 48
5.5 Linked List Process Hierarchy 49
5.6 Linked List Unit 0 DFD 50
5.7 Linked List Unit 0 CFD 50
5.8 Linked List Unit 1 DFD 52
5.9 Linked List Unit 1 CFD 52
5.10 Linked List Unit 2 packet example 53
5.11 Linked List Unit 2 DFD 53
5.12 Linked List Unit 2 CFD 54

5.13 Linked List Unit 3 DFD 55
5.14 Linked List Unit 3 CFD 55
5.15 Linked List Unit 4 DFD 58
5.16 Linked List Unit 4 CFD 58

6.1 Simulation Environment Context Diagram 61
6.2 Simulation Burst Read to Burst Write 62
6.3 Real Simulation Burst Read to Burst Write 63

A.1 Hatley and Pirbhai Elements and Symbols 67

B.1 Polling-Based Transfer 69
B.2 Interrupt Based Transfer 69
B.3 DMA Based Transfer-Initial DMA Operation 70
B.4 DMA Based Transfer-DMA Transfer 70

C.1 Burst Write State Transforms to Burst Read State 71
C.2 Burst Write State Transforms to Sequential-single State 71

List of Tables

3.1 Testing Cases 25
3.2 Subunit "Manage Counter" Input Signals 26

5.1 Linked List Unit Top Level Input and Output 51
5.2 Linked List Unit 2.1 Input and Output 54
5.3 Linked List Unit 2.2 Input and Output 54
5.4 Linked List Unit 2.3 Input and Output 55
5.5 Linked List Unit 3 Input and Output 56
5.6 Linked List Unit 3 Input and Output 56
5.7 Linked List Unit 3 Input and Output 57
5.8 Linked List Unit 4 Input and Output 57
5.9 Linked List Unit 4 Input and Output 59

1 Introduction

1.1 Project Motivation

In the last ten years, with the development of consumer electronics technology and the continuous dropping prices, people's experiences for the entertainments at home have been changed greatly. People are enjoying the revolution and asking for more. The speed of the change nowadays is even faster than ever in order to fulfill the consumer's desire. The time line for a company to release a new product becomes shorter and shorter. The consumer electronic products are smarter and smaller. At the same time, more and more people choose to own a car as their main vehicle, especially in the fast developing countries. This leads to people spending more time in their cars. They are eager to enjoy the same experiences as they have at home wherever they happen to be. Along with the progress on the consumer electronics, the In-car entertainments have also come a long way from the simple AM/FM radio and tape to more diversity functions. This situation can't be ignored by the car manufacturers and suppliers. At first, it's expensive to add new entertainments into the car. However, with the rapidly falling prices of the consumer electronics and new semiconductor technologies, a similar revolution is taken place in the automotive entertainment markets. Of course, car is one of them. The car entertainments not only include audio, but also video entertainment. The quality of car entertainment experiences are increased dramatically with the adoption of digital AM/FM radio, CD/DVD playback interface, gaming, hand's free telephony, navigation, rear-seat entertainment (including video) and personal portable devices, like USB. In-Car Entertainment System is developed and renewing to behave and control the entertainments in car. In-Car Entertainment System is an integrated embedded system combined different entertainment devices in a car. The development of integrated solutions and corresponding devices makes the adding of new functions more advanced and sophisticated.

However, things are not as easy as we expected. The data requirements for the In-Car Entertainment System are rising when integrating more devices. More and more entertainment forms lead to a diversity of data formats. These data formats are required to be processed and communicated on time to fulfill the passenger's needs. Originally, data computation and data communication tasks are all taken by the microprocessor, which is known as the Car Entertainment Processor in the In-Car Entertainment System. Both of the computation and communication tasks occupy lots of resources from the microprocessor. This greatly increases the load to the microprocessor and may cause errors when the microprocessor is not able to handle all the tasks. Heat problem is also very important for the microprocessor and the entertainment system. It's easy to be overheated when the microprocessor is fully running. The microprocessor can be instability, as a result. Moreover, more electronic functions cause more

power consumption and heat dissipation. Considering the limited space in a car and the spare parts which dissipate large amount of heat, like engine, the heat environment for the In-Car Entertainment System and for its microprocessor, is very crucial. Another problem causing situation is the long lifetime of a car, which is a very basic and important quality requirement for the car manufacturers. This also requires that the In-Car Entertainment System should be reliable enough for the long time operation. Therefore, a method has to be developed to offload the microprocessor from handling both the computation task and the communication task at the same time. One option is to design another functional unit to control the communication task instead of the microprocessor.

1.2 Project Goals

The project is to design a DMA Controller which is suitable for the In-Car Entertainment System. The YAD-C DMA Controller design project is a continuous project in NXP Semiconductors. This thesis project is the third stage of the whole project. The first step of this project is to test and analyze the existing results. After finding out the limitations of the existing results, possible ways for improvements should be proposed. Based on the first step, the functions of Core Unit should be re-specified and implemented using Hatley and Pirbhai Methodology in the second step to make sure the Core Unit can perform all the required transfer functions correctly. Simulation and synthesis should be done to check the feasibility of the implementation. In the third step, Linked List transfer function needs to specified.

1.3 Results and Contribution

The results of this thesis are: 1) A test principle has been investigated to test the functions of data transfer. 2) New counter management unit has been specified and implemented. 3) New State Machines of the Core Unit have been designed correctly to control the data transfer processes and the corresponding behaviors, like loading the required data. 4) The Core Unit is able to perform the sequential-single transfer and burst transfer functions with wait states insertion correctly according to the simulation results. 5) The Linked List Unit is specified according to the Hatley and Pirbhai methodology. 6) The synthesis result of one Core Unit Cell is 3k equivalent gates.

The proposed test principle has included all the possible situations for the data transfer with wait states insertion. Thus it can be used in the simulation of future implementations to test the functions of the newly implemented data transfer types and the corresponding function units, such as Linked List transfer and Linked List Unit. The Core Unit Cell is easily increased or decreased, so the DMA Controller can easily be adjusted in different application environments and different projects in NXP Semiconductors.

1.4 Thesis Organization

The rest parts of the thesis are organized in the following way:

Chapter 2 introduces the background of the data communication and DMA Controller. Three methods of data streaming are presented and compared. A general DMA Controller is described here. Two currently used DMA Controllers in NXP Semiconductors are introduced and the problems of these DMA Controller are discussed.

Chapter 3 gives an introduction of the existing results. A test principle is investigated to test the existing results. The analysis of the existing results is discussed and possible approaches for improvements are given.

Chapter 4 describes the specification and implementation of the Core Unit. A structure of the DMA Controller is discussed and the function requirements are presented during the Conceptual Architecture design. In the Logical Architecture design, the Core Unit is specified and fully implemented according to the analysis in Chapter 3.

Chapter 5 describes the Linked List Unit specification according to the principles of Hatley and Pirbhai Method. Linked List Unit is specified into 4 subunits and some of the subunits are further divided. Process Specifications for Linked List Unit and the subunits are also discussed.

Chapter 6 discusses the simulation and synthesis result of the Core Unit. One use case is chosen to explain the simulation result.

Chapter 7 concludes the results of the thesis project. Recommendations for future work are also given.

2 Background

This chapter introduces the background of the data communication and DMA Controller. Section 2.1 focuses on the three methods of data streaming. These methods are compared based on the consumed clock cycles to transfer one data elements. In Section 2.2, the structure and functions of a general DMA Controller are introduced. The basic operations of a DMA Controller are also presented. Based on the general module, Section 2.3 analyzes two currently used DMA Controllers in NXP Semiconductors. The problems of these DMA Controllers are discussed in the end.

2.1 Data Communication of the Design ICs

Figure 2.1 illustrates a modern In-Car Entertainment System:

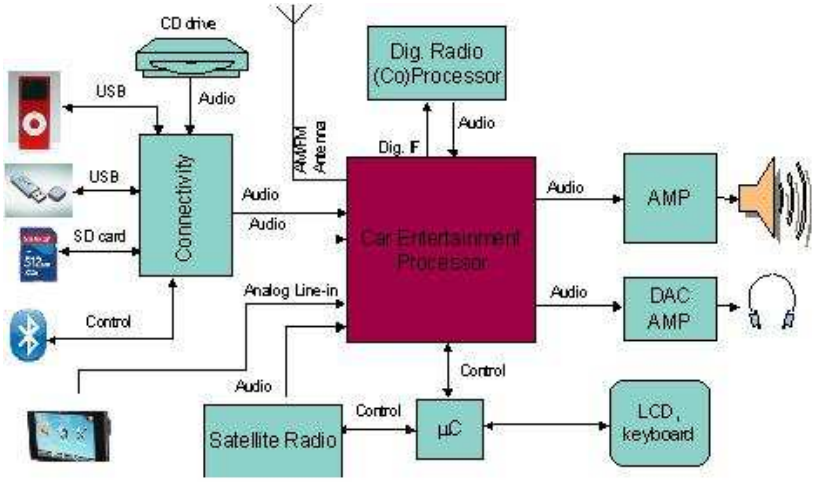

Figure 2.1: In-Car Entertainment System

Typically today in a head-unit of a car, there is the entertainment box containing several processing units (ICs) and different hooks towards external sources and sinks of entertainment.

There is the traditional AM/FM reception path from the car antenna. Other radio inputs can come from satellites in high-end car systems. Audio and video inputs generally come from either CD/DVD drive inputs, but can also come from connectivity devices, such as USB (from iPod, for instance), SD-Card. Navigation is nowadays an essential component in mid-end and high-end cars, and even some low-end cars. So are the external display outputs. Some car entertainment systems support external

Digital-to-Analog convertors for audio sinks near the amplifiers next to the speakers. Others expect analog input directly to the amplifiers.

The heart of the processing to be done on these radio, audio and video inputs is what is know as the Car Entertainment Processor. It is responsible for doing all kinds of processing from these sources. Equally important, they are also responsible to interface with the different sources and sinks mentioned above, each of these sources and sinks having their own interfaces and protocols running on these interfaces.

Finally, the "head" of the entertainment box is the micro-controller, that is responsible to communicate with external control buses in the car (like CAN bus, MOST bus for data), as well as for power and reset behavior of the entertainment box. It is also responsible for controlling and providing commands to the different ICs in the entertainment box, including to the Car Entertainment Processor.

2.1.1 Data Communication and Computation of the Design ICs

The car entertainment system typically processes the input signals or data and generates output signals or data. During its execution, the microprocessor not only executes the computation on the inputs, but also transfers the executed inputs to the output interface. Therefore, the function of the car entertainment system can be divided into two domains. One is computation, the other one is communication domain. The process is showed in the following Figure 2.2.

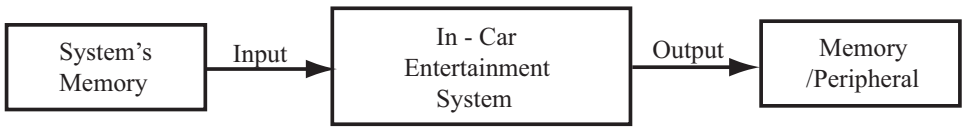

Figure 2.2: Simplified In Car Entertainment System

The data format varies for different entertainment functions. In order to execute different data formats in car entertainment system, there are a pair of corresponding Input and Output Interfaces. The Input Interfaces are responsible to convert a particular input data format into a standard format that can be processed by the car entertainment system. And the Output Interfaces are used to convert the standard format to an output data format which can be recognized by the external display device. The process is showed in Figure 2.3.

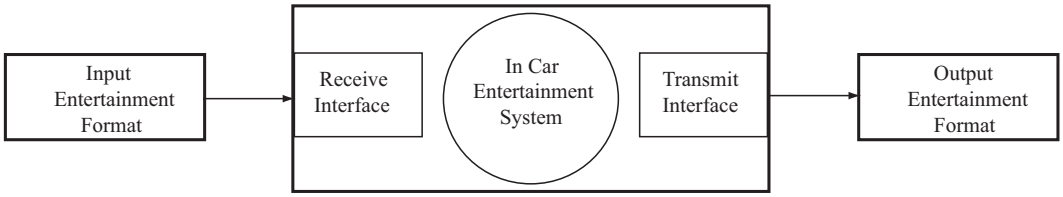

Figure 2.3: Single Entertainment Format

With the development of entertainment devices, people have more entertainment demands for In-car experiences. And more entertainment functions are also developed to the market, such as digital radio. When transforming different input formats at the same time, theoretically, each input signal can be transferred to any of the output interfaces. (See Figure 2.4, the input of FM/AM radio signal can be transfer to any of the four output interface.) Therefore, when different entertainment processes are executing, the car entertainment system has to handle module format transformation while transfer the concerned input signals to the corresponding Output Interface. Thus, the car entertainment system must spend a large number of overhead on both computation task and communication task.

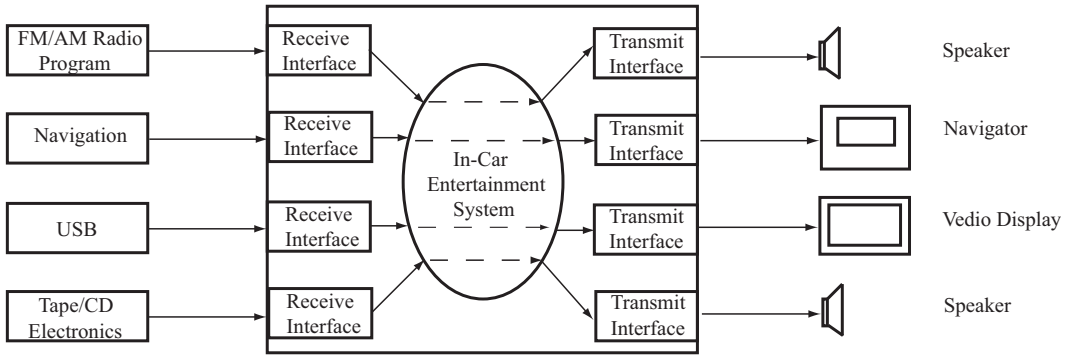

Figure 2.4: Multiple Entertainment Format

To deal with the situation above, one of the methods that nowadays industry adopts is to separate the communication domains from the computation module. Seen in Figure 2.5. A large number of overhead on the communication domain can be released and spend on the computation execution. The car entertainment system with a separated communication module has less area, consumed less power and also obtain high performance, because the system can do the data transfer while do the computation work at the same time.

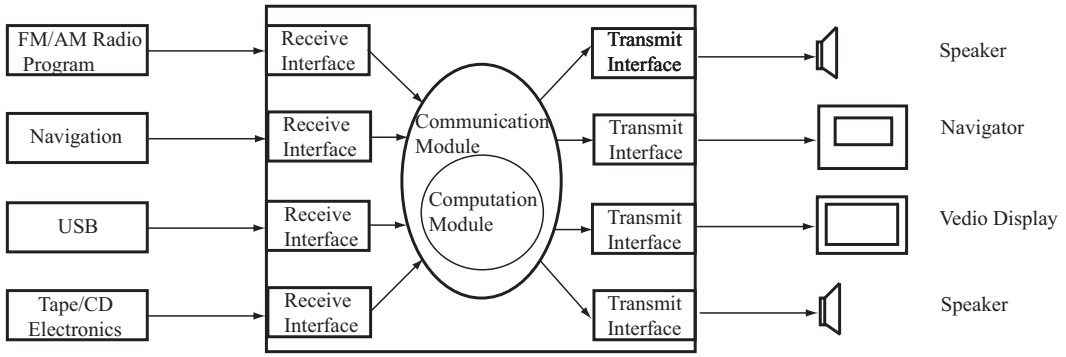

Figure 2.5: Multiple Format Separate Module

The concept of DMA Controller is developed. 'DMA' means 'Direct Memory Access'. So the DMA Controller is use to control the communication work instead of the microprocessor. And the DMA Controller is able to read or write the memory directly

without the interference of the microprocessor. This concept is widely used in the industry nowadays. And the Car Entertainment Business Line of NXP Semiconductor also wants to design a particular DMA Controller which is able to fulfill the increasing challenge along with the fast development of the In-Car Entertainment system.

2.1.2 Methods of data Streaming computation module

The data being fed in to an in-computation module as described in the former subsection is mostly streaming in nature. Streaming means there are real-time constraints when data has to arrive and sent out. For example, speech is typically constraint on 8 kHz, uncompressed audio on 44.1 kHz or 48 kHz, etc. It is the job of the "communication" module" in figures above to meet these constraints of a stream of data. Looking at the past and present embedded system, there are three methods to stream data. They are Polling-based transfer, Interrupt-based transfer, DMA-based transfer.

Polling-Based Transfer

A Polling-based transfer continuously polls or checks the peripheral status whether it is ready to transmit or receive data or not. In the polling-based transfer, when the peripheral is ready, the related flag in one of its status registers is set. In order to find out this new status, the computation module should continually checks the status registers of the peripheral in tight polling loops. If the status shows the peripheral is ready to transmit/receive data, the computation module should drop the loops immediately and start to perform a data transfer. After transferring the data item, the computation module computes new data. The microprocessor starts a new loop.

The context diagram of the polling-based system is showed in Figure 2.6.

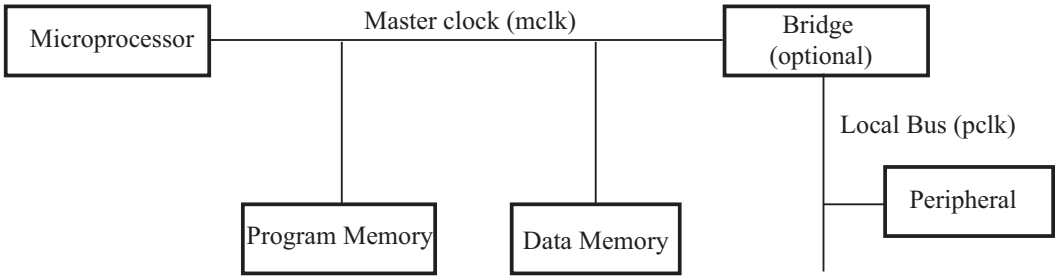

Figure 2.6: Polling based data transfer

The polling-based transfer is simple and requires less hardware. But from the description of the execution, it is clear that whenever data are to be received or transmitted, the status of the peripheral have to be checked first. This checking process can consume a lot of loop executions; therefore take a large amount of time. And during the checking execution, the computation module is not able to do anything else.

In order to understand this transfer better, a simulation is finished. The environment of the simulation is Advanced High-performance Bus (AHB) architecture compliant with the I2C peripherals. The data element is transferred from I2C peripheral 0 to I2C peripheral 2.

From the simulation, the processing time of status checking execution (one checking loop) was measured, as well as data transfer time to/from the receiving/sending

peripheral for one data element. And the total cycles that consumed during execution is 93 master bus cycles. (1 master bus cycle = 24 ns, 1 local bus cycle =15.8 ns) The simulation waveform can be seen in Figure B.1, Appendix B.

Interrupt-Based Transfer

An Interrupt-based transfer is to stop the current process of the computation module with an interrupt event so that the computation module is able to participate in another task, indicated by the event. In data handling, a peripheral interrupt is used to handle the data transfer task with the cooperation from interrupt controller. The Interrupt-based system context Diagram can be seen in Figure 2.7.

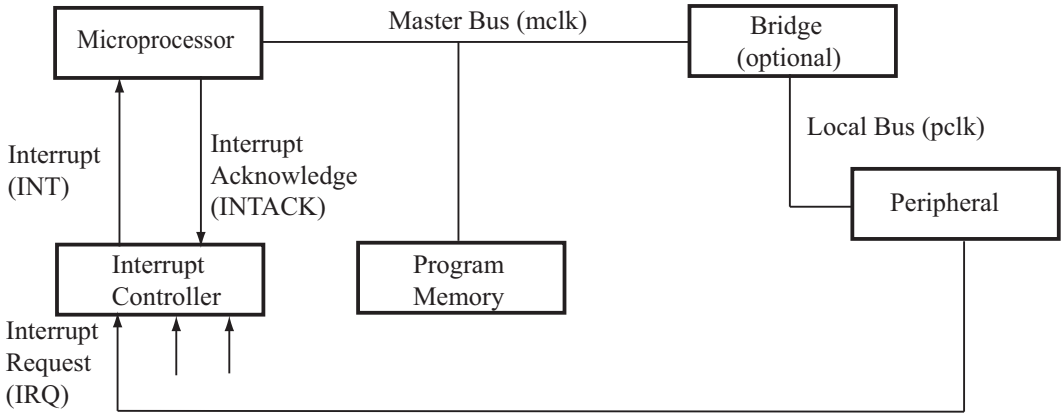

Figure 2.7: Interrupt based data transfer

When the peripheral indicate that it is ready for transmit/receive data, the concerned peripheral sends Interrupt Request (IRQ) to the Interrupt Controller which converts it into the corresponding Interrupt (INT) Vector. The vector is stored in the controller so that the computation module can read directly. The controller raises the INT signal to computation module to ask for service. On receiving the INT signal, the computation module reacts to the controller with an Interrupt Acknowledge signal to start the interrupt service. During the interrupt service, the computation module will do the following operations: First, the computation module checks which device gives the interrupt. After the sending peripheral is found, the computation module checks the status that caused the interrupt. If the status is ready for transferring data, the computation module starts the processes that are related to the peripheral and start a data transfer. At the end of the data transfer, the computation module clears the interrupt and resumes the previous process.

The interrupt-based transfer is efficient, because the processing operation of microprocessor is only suspended during the data transfer. And it relieves the computation module from having to continually poll for the peripheral status. But since the interrupt function requires a particular hardware, which is called Interrupt Controller, overhead of interrupt must be accepted. If it takes a long time for the computation module to perform data transfer, other processes have to wait until the task is finished.

In order to understand this transfer better, a simulation on executing the ISR is done. The environment of the simulation is Advanced High-performance Bus (AHB) architecture compliant with the I2C peripherals. The data element is transferred from

I2C peripheral 0 to I2C peripheral 2.

During the simulation, one data element was performed during the simulation. And it takes 217 master bus cycles to run the checking process. The data transfer of one element consumes 40 master bus cycles. Therefore, it totally costs 257 master bus cycles (around 260 master bus cycles) to execute the ISR for one data element. The simulation waveform can be seen in Figure B.2, Appendix B.

DMA-Based Transfer

Direct Memory Access (DMA) is a transfer that allows devices to transfer data from/to system memory to/from system peripheral automatically without intervention of the computation module. When a number of data elements need to be transferred from data memory, the computation module initiates the DMA Controller and grants the bus access to the DMA Controller. The DMA Controller starts a data transfer over the bus system. By the end of the transfer, DMA Controller interrupts the computation module to inform the completion of the data transfer task. The computation module takes control of the bus access from the DMA Controller. The block diagram of the DMA-based transfer is shown in Figure 2.8.

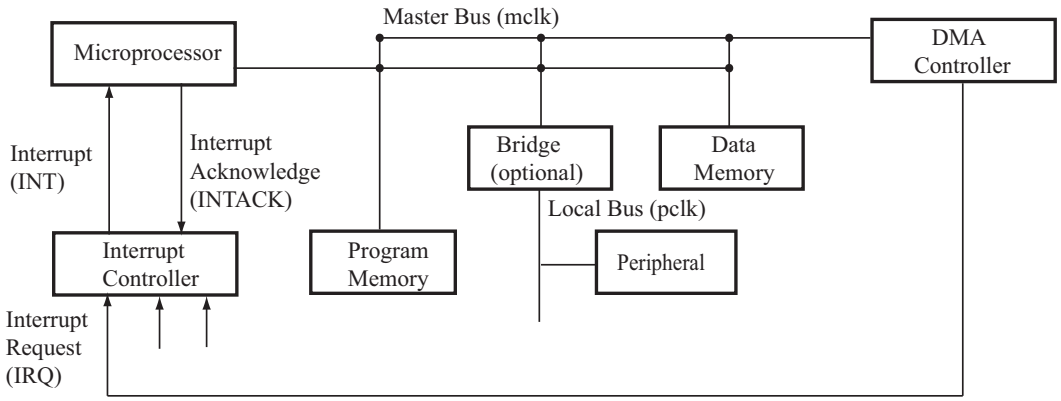

Figure 2.8: DMA based data transfer

From the description above, DMA Controller is able to offload the computation module. This provides the best efficient performance since the input and output processing can happen in parallel with other executions. But at the same time, the system becomes more complex.

During the simulation on DMA-based transfer, the master bus cycles that are consumed in the execution are measured and calculated. The environment of the simulation is Advanced High-performance Bus (AHB) architecture compliant with the I2C peripherals. The data element is transferred from I2C peripheral 0 to I2C peripheral 2.

It takes the computation module around 1650 master bus cycles to initiate DMA Controller, and 20 master bus cycles to transfer one data element. After the data transfer, the hardware interrupt which is handled by the computation module costs totally 273 master bus cycles, which contains 173 cycles to receive the ISR and 100 cycles to execute the ISR. Therefore, the whole operation consumes 1943 master bus cycles (around 2000 cycles). The simulation waveforms of the DMA initialization process and DMA transfer process can be seen in Figure B.3 and Figure B.4, respectively in

Appendix B.

2.1.3 Comparisons of the Three Methods

The clock cycles of the above simulations can be seen in Table 2.1.

Master Cycles	Polling	Interrupt	DMA
1 data element	93 master cycles per 1 polling loop	257 master cycles	1943 master cycles
10 data elements	930 master cycles per 10 polling loops	2570 master cycles	2123 master cycles

Table 2.1 Master Cycles Consumed by Data Transfer Methods

It's easy to see the Polling-based transfer is the worst among all the transfers. Because when checking the devices before data are ready to transfer, a large number of loops may be executed. And the computation module can only busy waiting and do nothing else. This is inefficient especially for the real time embedded system which is applicable in many designs on the market.

Comparing the interrupt-based and DMA-based transfers, one attractive advantage of the DMA-based transfer is the computation module can be offloaded. For the clock cycles that are consumed in execution, 10 elements were taken as an example. The total cycles for the interrupt-based transfer are 257*10=2570 master bus cycles. And the DMA-based transfer costs 1650+20*10+273=2123 master bus cycles. So the DMA-based transfer takes a large advantage of the consumed cycles, especially when the number of data elements to be transferred is large than 10.

The size of the current DMA Controller developed by NXP itself is only 30k gates. Compared to the computation module, which is 70k gates, the power consumption of the DMA Controller is much less than the computation module when doing the data transfers.

Using the DMA-based transfer, a DMA Controller, which leads to extra size and power consumption for the system, has to be introduced into the system. When there are a large number of data elements, considering the less power consumption compared with the computation module and the advantage of offloading the computation module, the DMA-based transfer is the best choice among all the transfers. Otherwise, If the amount of data elements is small (less than 10), it won't be beneficial for the extra size. In this situation, the interrupt-based transfer can be adopted.

2.2 Introduction of DMA Controller

2.2.1 Architecture of A General DMA Controller

2.2.1.1 Principles of DMA Controller

Stages of DMA Transfer

Similar to Interrupt-based transfer, the whole procedure of DMA transfer is separated into several stages. Before DMA transfer starts, users need to program DMA Controller based on the demand of the system. This pre-programmed information contains the size of the data, the source address, the destination address and some other configuration signals. This process is called initialization of DMA Controller. When the peripheral is ready, the DMA Controller can start to apply for the DMA transfer.

In the application stage, for a single-layer bus system, when the peripheral requests data transfer, a request signal (DREQ) will be sent to DMA Controller. Right after the DMA Controller accepts this request, it'll send a bus request signal (BREQ) to the microprocessor to ask for the occupation to the bus system. Then in the response stage, the microprocessor checks at the end of each bus cycle to see whether the BREQ signal is valid. As soon as the BREQ signal is ready, and at the same time, the bus system is not locked by other masters, the microprocessor will response to the BREQ signal and release the bus system. Thus, the DMA Controller takes over the bus system and becomes the control unit of the system. The third stage is to perform data transfer. The DMA Controller replies to the peripheral's request to recognize it as the selected peripheral for the data transfer task. On the other hand, using its bus control authority, the DMA Controller sends the address signal to the memory, the read/write control signal to both the memory and the peripheral, to control the data transfer and transfer path. After the data transfer is finished, the DMA Controller turns into the last stage, after-transfer stage. According to the pre-programmed information, when the data elements are all transferred, the DMA Controller generates a "'transfer complete"' signal for the peripheral. After receiving this signal, the peripheral cancels the DREQ signal. In consequence, the BREQ signal becomes invalid and the DMA Controller releases the bus system. Then the microprocessor takes over the bus again. This is the whole procedure for one DMA transfer.

Applications of DMA Transfer

DMA method is used to enhance the data throughput of the system. It is mainly adopted in the data transfer system which requires high speed and processes large number of data batch, such as disk accessing, graphic processing. But compared with other transfer method, DMA transfer utilizes extra hardware units to take over the same function from software. This leads to an increase in the hardware flexibility and cost of the system. In the single-layer Bus system, the DMA Controller takes over the bus system from microprocessor. Thus, if there are some important interrupt requests arriving during this period, the microprocessor will not be able to response in time. Therefore, DMA method is not necessary for single-layer bus system. In multi-layer bus system, when the DMA Controller and the microprocessor are masters of two different layers respectively, both of the DMA Controller and the microprocessor can access to the bus system at the same time. In this case, the utilization of the DMA Controller makes the system more efficient.

2.2.1.2 Components of A General DMA Controller

A DMA Controller can be seen as an interface block among microprocessor, peripherals and bus system. This interface block is a circuit of DMA transfer mechanism built on an interrupt block. Therefore, the DMA Controller consists of an interrupt part and a

DMA part. Figure 2.9 shows a simple structure of the DMA Controller.

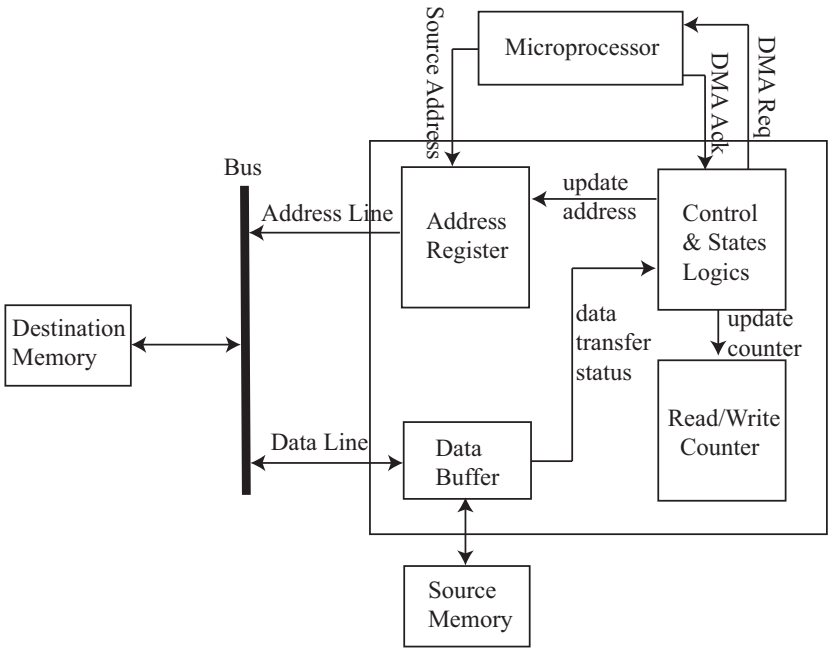

Figure 2.9: Simple Structure of a DMA Controller

Address Register

Address Register is used to store the addresses of the required data elements. Before DMA transfer, the microprocessor must send the source address to this Address Register. During DMA transfer, at the end of each single data element transfer, the value of the register will increase '1'. Thus, the addresses are provided in a incrementation form.

Read/Write Counter

Read/Write Counter(R/W Counter) is used to record the length of the transferred data elements. The initial value of the counter is also set by the microprocessor before the DMA transfer. The value equals to the length of the required data elements. During DMA transfer, the value of the counter decrease "'1"' right after one data element is transferred. So the R/W Counter performs in a decrement form. When the counter turns to "'0"', the DMA transfer of the current data group is finished and the DMA Controller send an interrupt signal to the microprocessor.

Data Buffer

The Data Buffer is used to temporally store one data elements. When the data element is imported, it is transferred first from the source memory to the buffer, then from the buffer to the system bus and finally the destination memory and vice versa.

Control and State Logics

This part of the circuit is composed of Control and Timing Logics and Status Flag. It is used to modify the value of Address Register and R/W Counter, set read or write process, and so on.

2.2.2 Function Descriptions of A DMA Controller

Basic functions of DMA Controller

According to the description of DMA transfer, some basic functions of DMA Controller can be summarized. A DMA Controller can be programmed by microprocessor in order to set the initialization information. A DMA request signal is sent to microprocessor. After microprocessor accepts the DMA request, the DMA Controller starts DMA operation. During the DMA operation period, the DMA Controller can send address signals to address lines, and send read/write control signal to control lines. The size of data elements is also controllable by the DMA Controller. When the data transfer task is finished, the DMA Controller is able to send a complete signal and release the bus.

Except the above basic functions, there are other functions added into some DMA Controllers, for instance, generating interrupt request when the DMA transfer is finished; accepting new source address and the data size information in case the DMA Controller retransfers the previous data blocks again, or connects the finished data block with other unrequested data blocks together in the following DMA transfer; generating 2 memory addresses so as to achieve the transfer between memory and memory.

Functional Modes of DMA Controller

DMA Controller is design as a data transfer interface block between two memory blocks. Compared with other I/O interface device, the DMA Controller is able to perform either as the system controller or as a peripheral unit which is controlled by microprocessor. Therefore, there are two functional modes for the DMA Controller existing in the system. They are Active Mode and Passive Mode.

Active Mode

The active mode is the process that performs the work needed to move the data. The active mode allows the DMA Controller can either be a data source unit or a data destination unit. When the DMA Controller performs as a source unit of the data transfer, data elements are written by the DMA Controller to the memory or peripherals. When performing as a destination unit, the DMA Controller reads the data elements from the I/O unit. Therefore, the DMA Controller in active mode is a bus master. During the procedure of DMA transfer, the DMA Controller sends a 'load data read' signal to trigger read operation and reads data elements from the memory and then writes the data to the destination peripheral. Then the DMA Controller sends a 'load data write' signal to trigger write operation and write the data elements which is read just now to the destination peripheral.

Passive Mode

The passive mode is the process whose memories are written to or read from by the processes in the active mode. The passive mode is not finished until all the data is written to the DMA Controller. Before the DMA transfer, the DMA Controller is controlled by the microprocessor. When the request signal is accepted, the microprocessor pre-programs the DMA Controller. A descriptor is sent to the DMA Controller. This descriptor consists of the initialization information, such as the source and destination address, the size of data, and some other configuration information. When the DMA transfer is finished, the DMA Controller halts in the passive mode and performs as a bus slave.

2.2.3 Basic Operation Modes and Transfer types

When the microprocessor accepts the DMA request, the DMA Controller can operates in three basic modes: cycle Stealing, interleaved DMA, suspend microprocessor access. Each of them has its advantages and disadvantages.

2.2.3.1 Basic Operation Modes of DMA Controller

Cycle Stealing DMA

Cycle stealing means the DMA Controller 'steals' one single microprocessor clock cycle from the microprocessor to perform DMA transfer. The system bus is needed both by the microprocessor and the DMA Controller when they are executing instructions. In a single-layer bus system, if the DMA Controller requests to access the system bus, while the microprocessor is accessing the bus at the same time, executions of instructions of the microprocessor are delayed and the DMA Controller takes over the bus control to transfer a single byte or word. Then the DMA Controller becomes Idle again and waits for another request from the peripheral. In fact, the DMA operation can be seen as an insertion into the microprocessor bus cycles. Because of the insertion, the cycle stealing DMA mode decreases the operation speed of the microprocessor. And another limitation of cycle stealing DMA is only one byte or word data elements can be transfer during one DMA operation. And for each cycle stealing, the DMA Controller has to request the authority of system bus, set up bus control and release the bus to the microprocessor. Thus, it takes more than one microprocessor clock cycles to finish the data transfer compared with one clock cycle by microprocessor itself. Therefore, it's not suitable for the transaction requesting high transfer speed and large amount of data.

Interleaved DMA

The interleaved DMA accesses the system bus only when the microprocessor is not using it, for instance, performing an ALU operation or incrementing a program counter. The DMA Controller chip identifies these cycles and allows transfer of data between the memory and I/O device [3].

Suspending Microprocessor

In the suspending microprocessor method, for the single-layer bus system, the DMA Controller takes over the authority of system bus from microprocessor to perform data transfer. During DMA transfer, the microprocessor can execute the other operations which do not need to access the bus. Only when the DMA operation is finished, the microprocessor can access the system bus again. This method is popular with microprocessors [3]. Suspending microprocessor will slow down the speed of microprocessor, but not as much as the clock cycles microprocessor consumed in data transfer.

2.2.3.2 Transfer types of DMA Controller

There are three transfer types during the DMA transfer. They are single word transfer type, block transfer type and demand transfer type.

Single Transfer Type

In single transfer mode, the DMA Controller is programmed to perform one transfer only. Therefore, the single transfer is an inconsequential transfer, which means the address and the control signals are not related to the signals of previous data transfer. The read/write counter will decrease by one and the address will increase following each transfer. After the transfer, the DMA Controller is halted and release the bus. When next word of DMA transfer is needed, the DMA Controller has to request again.

Block Transfer Type

When the block transfer starts, an entire data block will be transferred. The DMA Controller disables itself when the transfer is complete. During this transfer, even though the DMA request turns to inactive, the DMA Controller still controls the system bus and halts the DMA operation at the same time. After the DMA request becomes active, the DMA operation continues making transfer. Block transfer is sequential. The address signal is based on the previous transfer. The value of the address is the previous address plus the size (by bytes). And the control signal is identical to the previous transfer. There are two ways of block transfer type: sequential single transfer and burst transfer.

In sequential single transfer, the DMA Controller transfers the block data elements one by one. That means, right after one data element is read from the source, the DMA Controller performs the write operation to send this data element to its destination. And the read/write counter decreases one after each transfer. When the counter finally becomes 0, the transfer is complete. Then the microprocessor takes over the system bus.

In burst transfer mode, the DMA Controller sends a burst of data elements to destination repeatedly. Compared to the sequential single transfer, the DMA Controller won't perform the write execution until all the data elements in the burst are read, and vice versa. There are burst read/write counter to record the number of transferred data elements, and will increase after one element is transferred. When the burst read counter reaches the size of the burst, the DMA Controller stops the read process to write the burst to its destination. After all the data elements in the burst are finished writing, the next burst can start to be read. The data elements inside one burst are transferred sequentially. And between each burst, the microprocessor is able to interleave with the DMA operation. This leads to a capacity reduce of the microprocessor. But the microprocessor does not need to be halted completely. The burst mode disables itself when the transfer of the whole block is finished.

Demand Transfer Type

Demand transfer type is similar to the block transfer type, except that after each data element is transferred, the DMA Controller checks whether the DMA request is valid. Therefore, it's a way of polling as well. If the request goes inactive, the DMA Controller will release the system bus. Otherwise, the DMA Controller continues to make transfer until the block is completely transferred. And in this case, the DMA Controller performs as the block transfer type. Therefore, by controlling the active or inactive of the DMA request signal, a data block can be divided into several parts, and transferred separately each time period sequentially when the I/O device is ready.

2.2.4 Applications and Trends of DMA Controller

DMA Controller is responsible for the data transfer between memory and memory or peripherals without interference from the microprocessor. This application not only increases the data transfer rate between memories, but also frees the microprocessor from large amounts of data transfer, collecting scattering data elements, visiting slow devices and hence makes the microprocessor become more efficiency.

The DMA Controller nowadays is becoming more sophisticated and intelligent. The operation of normal DMA controller involves the operation from the microprocessor. DMA Controlled data transfer has to be established entirely by instructions of the microprocessor. And the DMA Controller is adequate for the requirements of microcomputers and small computers. As far as the large computer or more sophisticated embedded systems, their CPU or microprocessor is extremely powerful and the system bus usually attaches with a large number and varieties of peripherals. In this case, a separate DMA Controller for each device is uneconomical [4]. Therefore, device called Input/Output Processor (IOP) is developed.

During the operation, the microprocessor or CPU is responsible for initiating the input/output program. And then the IOP fetches and executes its instruction to perform the data transfer between peripherals and memories. At the same time, the microprocessor or CPU can process data which is needed in the data computation tasks. IOP is also need to structure the data from different peripherals or memories, because the data formats are different among different peripherals, and between the peripherals and the memories as well. For instance, when an IOP receives a 2 bytes input data, it must temporally store these data and pack the format to the memory format, which is 16 bytes. After data packing, the input data is able to be transferred to the memory. After data transfer task is finished, IOP informs CPU with an interrupt signal.

More sophisticated DMA Controller involves the scatter/gather function. This is also known as linked list transfer, whereby a DMA Controller is able to read/write data from/to separate locations in memories or peripherals. The transfer is arranged by a task list which is set in a descriptor. DMA Controller is initiated by the microprocessor according to the instructions in the descriptor. More details of the linked list transfer will be talked about later.

2.3 Limitations of Currently used DMA Controllers in NXP

Two kinds of DMA Controllers are adopted in current projects of Car Entertainment Business Line, NXP Semiconductors. Because of the confidential reasons, the names of the DMA Controller will not be indicated here. Instead, they are named as DMA Controller 1 and DMA Controller 2.

2.3.1 DMA Controller 1 used in NXP

The DMA Controller1 used in the current design is called Simple DMA Controller (SDMA) which is a central DMA Controller that connects to the Advanced High-performance Bus (AHB) and VPB system. A system with a central DMA Controller consists of a processor and a DMA Controller as a master on the bus; the peripheral

and the memory are slaves on the bus. The system context diagram is showed in Figure 2.10.

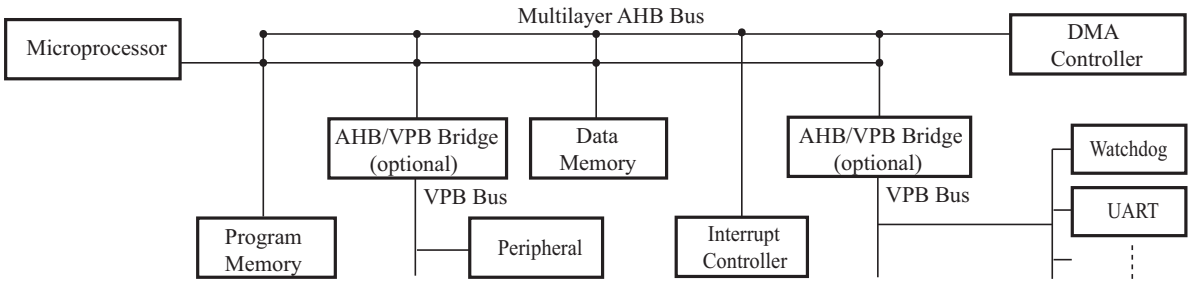

Figure 2.10: DMA Controller 1

The system contains four parts, see Figure 2.11 [5]. The VPB interface whereby the microprocessor can control the SDMA registers; The AHB interface whereby the SDMA performs this DMA functionality (data transfer), DMA channels; The flow control interface, connected to the DMA peripherals which triggers a DMA cycle; The external DMA enable interface and interrupt interface.

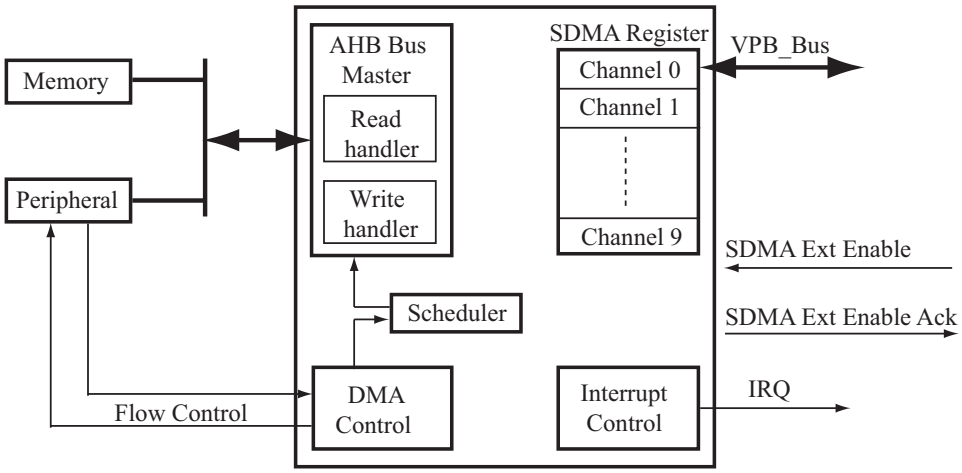

Figure 2.11: Simple DMA Controller

The SDMA Controller provides some following functionality: APB/VPB interface for sending control into from microprocessor to DMA; DMA cycles on the AHB bus; Supports byte, halfword and word transfers, and correctly aligned it over the AHB bus; Supports 31 peripherals for DMA flow control; Supports external enabling (SDMA Ext Enable pins) of the DMA channels, so other peripherals then the microprocessor can enable one or more DMA channels.

2.3.2 DMA Controller 2 used in NXP

The DMA Controller 2 is an Advanced Micro-controller Bus Architecture (AMBA) compliant System-on-Chip (SOC) peripheral. The DMA Controller is an AMBA AHB

module, and connects to the Advanced High-performance Bus(AHB). This DMA Controller contains two AHB masters. This enables, for instance, the DMA Controller to transfer data directly between the memory and the peripheral on AHB bus. The transactions which are occurred independently on AHB bus between the DMA Controller and the APB peripheral are also possible. The system context diagram can be seen in Figure 2.12.

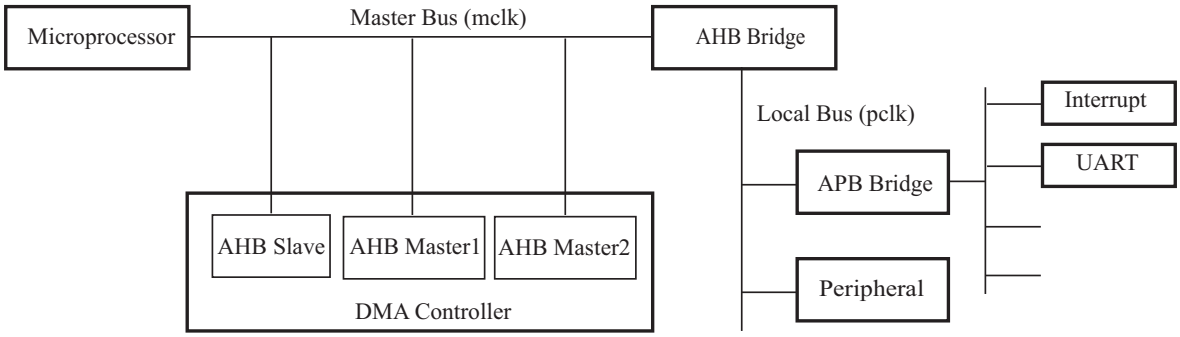

Figure 2.12: DMA Controller 2

The DMA Controller offers some following features: Compliance to the AMBA Specification for easy integration into SoC implementation; Eight DMA channels. Each channel can support a unidirectional transfer; Single DMA and burst DMA request signals. Each peripheral connected to the DMAC can assert either a burst DMA request or a single DMA request; The DMA burst size can be set by programming the DMAC; Scatter or gather DMA support through the use of static linked lists; Two AHB bus masters for transferring data. Use these interfaces to transfer data when a DMA request goes active.

2.3.3 Problems of these DMA Controllers

Using the DMA Controllers mentioned above, several disadvantages of those DMA Controllers were found by NXP engineers. The schedule algorithm used in current design is not flexible. The algorithm currently used, which is called Round Robin schedules all active channels in equal portions and order, treating all channels equally. After each DMA cycle, a channel switch takes place and the following active channel will be granted to the bus system. It's suitable for streaming data with a low latency and small message. If a large amount of data need to be transferred through bus system, the bus latency requires more clock cycles than channel latency.

The current DMA Controllers are also not flexible in the number of bus masters and channels. For the microprocessor DMA Controller, the number of bus masters for transferring data is fixed into two. The linked list transfer function provided by the DMA Controller 2 is static linked list. The future Network-on-Chip communication system requires the transaction of communication packets of variable size. Therefore, a support for the dynamic linked list transfer is needed.

2.4 Conclusions

Among three data streaming methods, the Polling-Based transfer has the worst performance. Compared with the Interrupt-Based transfer, the DMA-Based transfer is better when the number of requested data elements is larger than 10. Therefore, the DMA-Based transfer is the most suitable data streaming method towards the In-Car Entertainment System. The DMA Controller is designed to perform the DMA transfer functions. A DMA Controller can works in an active mode or a passive mode. When working in the active mode, there are three modes for the operation of the DMA Controller: Cycle Stealing DMA, Interleaved DMA and Suspending Microprocessor. Suspending Microprocessor is the most common mode to use. When the DMA Controller is triggered, there are three transfer types during DMA transfer: Single Transfer type, Block Transfer type and Demand Transfer type. Moreover, there are two Block Transfer modes: Sequential-single transfer and Burst transfer. The currently used DMA Controllers in NXP Semiconductors are not flexible adopted into the future development and research in NXP Semiconductors both in algorithm domain and architecture domain. Therefore, a newly designed DMA Controller is requested by NXP.

Problem Analysis of Existing Results

3

This chapter focuses on the introduction and analysis of the existing work and the possible approaches for improvements. In section 3.1, the existing results are introduced. Section 3.2 investigates a test principle, which is used to test the functions of the implementation. This principle is also adopted to test the implementation of thesis results. According to the simulation, the analysis of the existing results is discussed and possible approaches for improvements are given.

3.1 Introduction of Existing Results

The designed DMA Controller is named 'YAD-C', short for 'Yet Another DMA Controller'. There have been two stages of design before and this thesis project is the third stage. Throughout both stages, Hatley and Pirbhai methodology was used. A brief introduction of this methodology is given in Appendix A. In the first stage, the function requirements of the DMA Controller were specified. According to these requirements, the DMA Controller is divided into several function units, among which the Core Unit is the most important one. It is responsible to transfer data elements. Figure 3.1 is the Context Diagram of the Conceptual Architecture, which shows the relations between the DMA Controller and its surrounding function units in the system.

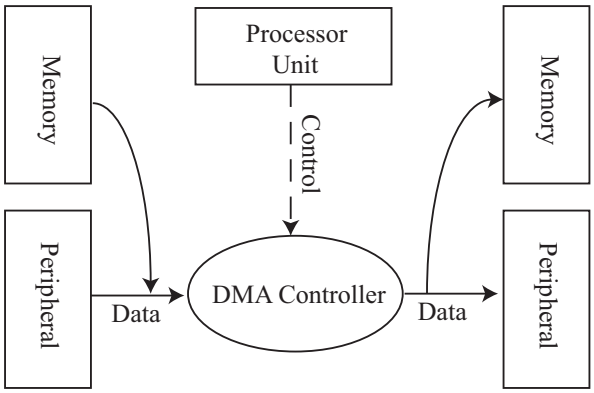

Figure 3.1: First Stage DMA Controller Context Diagram [1]

The Processor Unit triggers the DMA Controller by sending control signals; thereafter the DMA Controller starts working on data transfer task in the absence of interference from the Processor Unit. When the transfer task is finished, the DMA Controller inserts an interrupt signal to the Processor Unit to stop DMA operation. During DMA operation, the DMA Controller first reads the data from the source address, which is located in the system memory or other peripherals. Afterwards the data is written to

the memory or peripheral when the destination address is ready.

However, the Logical Specifications in this stage are not precise enough to describe the operations of the DMA Controller. So the second stage is started from the re-specification of the DMA Controller according to the Conceptual Architecture from the first stage. The DMA Controller is divided into 8 subprocesses based on the function requirements. Each subprocess takes responsibility for one particular function and would be designed as a subunit of the DMA Controller. Figure 3.2 shows the Data Flow Diagrams of the top level Logical Specification.

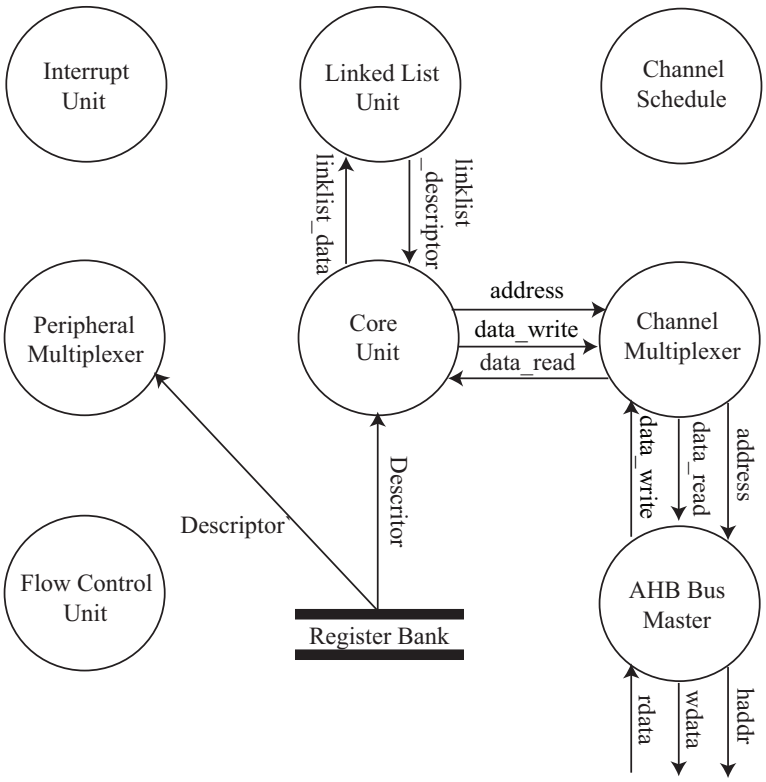

Figure 3.2: Second Stage Top Level Data Flow Diagram [2]

Based on the functional and operational descriptions, top level requirements are set in order to indicate the functions that are expected for the DMA Controller. YAD-C DMA Controller is designed to meet the requirements on: The DMA controller must be designed in flexible units, which can be modified and extended in future; The DMA controller could be used for Central DMA transfers; Interrupt generation when transfer completion, or bus error during reading or writing; Multiple logical DMA channels; Different ways for enabling a channel, which includes an enable bit put into a register of the register bank of the DMA controller, an external enable pin [2].

The Core Unit performs the data transfer task from the source to the destination. The AHB Master Unit controls the communication between the Core Unit and the AHB bus system. The Channel Multiplexer is used to assign a DMA channel to AHB Master Unit. The appropriate DMA channel for the Channel Multiplexer is determined by the Channel Scheduler. The Linked List Unit prepares the Linked List descriptor

which contains the control signals for a Linked List transfer and forwards it to the Core Unit. Then the Core Unit could perform the Linked List transfer. The Flow Control Unit controls handshake signals for the peripherals, which is used during data transfer between DMA controller and peripherals. The Peripheral Multiplexer is used to select a suitable flow control signal for the Core Unit when multiple peripherals require data transfer. The Interrupt Unit is responsible to send the interrupt signal to the Processor Unit when the DMA operation is finished. Finally, The Register Bank is the memory of the DMA Controller. After the specification, the Core Unit is further specified and implemented.

The Core Unit is specified into 2 subunits. One subunit is 'Prepare and Manage Transfer', which is responsible to fix the address and counter values. The other subunit is 'Perform Transfer', which is used to perform data transfer task. A finite state machine (FSM) is designed to control the read and write processes. The Core Unit is designed to support both the sequential-single and burst transfer with wait states insertion. The wait states are required because different components may execute at different clock frequencies. Wait states are used to extend the current state so that the data can get enough clock cycles to be transferred. However, wait states influence the performance of the microprocessor owing to the clock cycles wasted in the waiting processes. The functions that have been implemented are the sequential-single data transfer with wait states insertion, and burst transfer without wait states insertion. Therefore, to complete all the required data transfer functions of the Core Unit, the burst transfer with wait states insertion and the combined data transfer (sequential-single plus burst transfer) with wait states insertion still need to be implemented.

3.2 Problem Analysis of Existing Results

3.2.1 Test Principle Setup

Requirements Analysis

In order to fully test the required functions of the Core Unit, three factors must be considered: 1) The inserted location of the wait states. 2) The number of requested data elements.

The inserted location

According to the function requirements, the Core Unit must support both the sequential-single transfer and burst transfer with wait states insertion. For the sequential-single transfer, the wait states can be only inserted before the data element is read or written. But for the burst transfer, the situations are more complicated. Since one burst of data consists of four data elements, there are four possible locations to insert wait states. According to AHB protocol, the address phase of the data transfer comes one clock cycles before the data phase. Therefore, in a burst state, there are four address phases as well. Thus, these locations are: 1) before data transfer, which means the wait states are inserted before the address phase of the first data element. 2) Between the first and the second address phases. 3) Between the second and the third address phases. 4) Between the third and the fourth address phases. The examples can be seen in Figure 3.3.

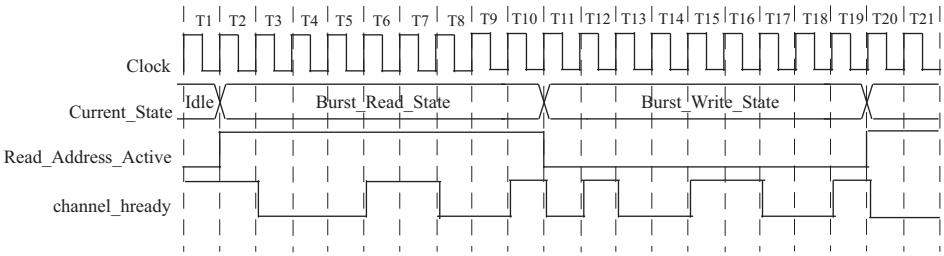

Figure 3.3: Location Examples of Wait States Insertion

The T2 period is the address phase of the first data element. And the T6 period is the address phase of the second data element. Thus, the wait states are inserted between the first address phase and the second address phase, which is the case two location. Other examples of inserted locations are shown in this figure as well.

The number of requested data elements

To make sure the functions are fully tested, the number of data elements is set from 0 to 23. Most of the cases are prime numbers. This is because of the requirements of Core Unit. The Core Unit is designed to perform the burst data transfer before the sequential-single transfer. Only when both transfer types are performed correctly, then we can say the design is correct. One burst consists of 4 data elements. Thus, the number of data elements for burst transfer must be a multiple of 4. In the case when both transfer types are required in a transfer task, the number of data elements should maximally 3 data elements larger than a multiple 4 number.

The maximum amount of data elements to test the function is set to 23. When doing the burst transfer, the randomly inserted wait states can influence the status of burst read and write states transition. In order to observe the transitions, two burst of data is required. So the test can be arranged to perform the transition from the first burst read state to the first burst write state. Then the first burst write state goes to the second burst read state. If the possible situations of wait state insertion can be arranged fully, two burst of data is enough to show the function. And the transfer should also be tested when there's no wait states occurs. Therefore, one burst of data is required to be transfer without wait state insertion. The sequential-single transfer after burst transfer might only perform 1 data element. Thus one burst of data with wait state insertion should be performed before the sequential-single transfer in case to the interaction between the states transition. Based on the above analysis, 23 data elements can fully test the data transfer function.

Test Principle Setup

The Test Principles can be concluded as:

1) There are four possible inserted locations. In order to test all the possible cases, first insert the wait states into one location, and change the insertion of other location. For example, first insert wait states into the first location, so for the rest of locations, there are 8 possible ways of insertion. For each location, the number of consumed clock cycles for one wait state can be more than one. So there are more than 8 cases of wait states insertion for this location. And then insert the wait states into the second location, and ignore the wait states insertion for the first location. This progress

continues till the fourth location.

2) For the number of the data elements, according to the analysis, different cases are chosen corresponding to the transfer types. And the details are listed in the Table 3.1. And for each test case, wait states are inserted as much complex as possible.

Transfer Type	Number of elements	Wait State
Burst	4,8,16,20	yes
Sequential-Single	0,1,2,3	0:no; Others:yes
Burst+Sequential-Single	5,6,7,17,18,19,21,22,23	yes

Table 3.1: Testing Cases

Different cases are chosen corresponding to the transfer types. The details are listed in Table 3.1. For each test case, wait states are inserted as much complex as possible. This is to emulate all the possible situations caused by wait states insertion. So when the transfer contains burst state, normally three wait states are inserted in one burst read or write process. And at least one wait state takes 2 clock cycles. The transfer without wait states needs to be tested as well. The same principle is also used in the simulation of this thesis project.

3.2.2 Simulation Results

The limitations of the existing work are the data transfer function was not fully accomplished. As mentioned in the previous section, he finished the sequential data transfer with wait states and burst transfer without wait states. But with the same codes, the functions which he had not accomplished, including burst data transfer and combined data transfer with wait states, performed incorrectly. When inserting wait states during burst transfer, problems happened either on the number of data elements transferred, or the values of the read and write data counters, which was responsible for counting the number of transferred data. By inserting wait states into a burst of data, the counter was not able to be synchronized with the data transfer. And the data elements in a data burst can't be fully transferred. For example, in the cases shown in Figure 3.4, only three data elements in the burst are written.

3.2.3 Problem Analysis

To analyze this problem, the flow diagram of this Core Unit was redrawn according to the control signal flow. Compared with the original flow diagram, it is clearer that there are two counter related processes. One is Core Unit 1 (Prepare and Manage Transfer), the other one is Core Unit 2.3 (Manage Counter). Core Unit 1 manages all the addresses and counters values according to the initialization information in the Register Bank. Core Unit 2.3 defines the increment, decrement of the counter. Instead of executing the increment and decrement operation, Core Unit 2.3 triggers Core Unit 1 to control the volume variation. Therefore, it takes 2 clock cycles to increment or decrement the counter after loading a new data. But when wait states are inserted into a data burst, the state machine which controls the data transfer might be unstable. Because the state machine, in this case, was controlled by the counter. Therefore, when inserting

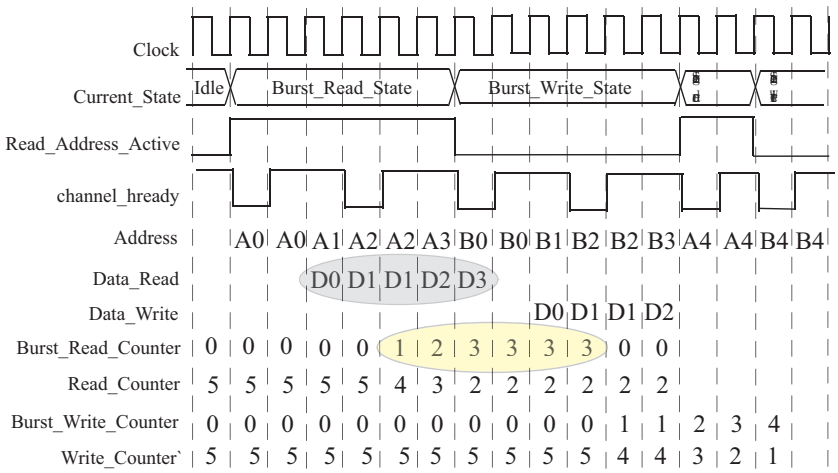

Figure 3.4: Example of Incorrect Burst Transfer

irregular wait states into a burst of data, the transferred data might not correspond with the counter. The number of transferred data elements in one burst fails to follow the setting according to the information in the descriptor. In some cases, only 3 data elements can be transferred. And in some other cases, more than 4 data elements are transferred. On the other hand, even the wrong data elements are transferred, the counter can't be synchronous to the data transfer process.

3.2.4 Possible Approaches for Improvements

According to the Hatley and Pirbhai design method, the design implementation starts from the Data and Control Flow Diagram. As mentioned above, the Core Unit 1 and Core Unit 2.3 are responsible for the counter management process.

The burst read and write counter is not decided by whether one data element transfer is finished or not. It's triggered by a burst counter control signal, 'increase burst read/write counter'. This burst counter control signal is generated by a subunit named 'Manage Counter' of the Core Unit 2. The 'Manage Counter' subunit defines when the counters should be increased and decreased. In the existing solusions, several signals are set to generate this trigger signal. These signals are listed in the Table 3.2:

Signal	Description
Channel hready	generate wait states
data read/write loaded	finish current data elements read or write
burst read/write counter	counts the number of transferred data elements in a burst
read/write counter	count the number of the data elements waiting for transfer
read address active	mark continuous read addresses for one read burst
data read/write memorized	memorize wait state

Table 3.2: Subunit "Manage Counter" Input Signals

'data read/write memorized' signals keep at '0' when there's no wait state coming. When wait states needed to be inserted into AHB bus to slow down the data transfer

speed, 'channel hready' signal will be used. In this case, to extend the current transferring data element, 'channel hready' signal is driven to '0'. Thus, on the next cycle, 'data read/write memorized' signals will becomes '1'. 'data read/write memorized' signal will returns to '0' 1 cycle after 'channel hready' signal returns to '1' again, which means wait states are finished. But the fact is there might be more than 1 wait state inserted consequently, for each time different number of wait states inserted, the values of the 'burst read/write counter' and 'read/write counter' may be different either. As well as signal 'data read/write loaded', because only after wait states finished, the current data element can then be transferred into the memory or the destination peripheral. So there are a group of different situations there. The existing results only include the case with 1 wait state and also failed to indicate all the conditions in 1 wait state insertion. Therefore, when the conditions came outside the box, there was an error happened. This led to incorrect value of the values of signal 'increase burst read/write counter' and 'read/write counter'. Consequently, when the wrong trigger signals were sent to Core Unit 1 to execute the counter increment or decrement processes, the values of the counters were incorrect too. Furthermore, the values of the counters were also the necessary conditions to the state machine which was responsible to perform data transfer in Core Unit 2. As a result, the data transfer task was mistaken as well. That's the reason when error happened, sometimes only 3 data elements were transferred as a burst and sometimes 5 data elements are transferred. Thus, it's clear that to fix the counter management process is the first task needed to be accomplished.

Besides the errors of the counter design, the mistakes may also happen in the unit which is responsible for performing the data transfer. According to the previous student's specification, the unit is Core Unit 2. One of the most important input signals of the state machine is burst counter, which included burst read counter and burst write counter. Because of the mistake of the counter design, nearly all the subunits of Core Unit 2 should be debugged. But the analysis is mainly focused on the state machine. Because generally the data transfer process can be divided into 2 states: read state and write state. Based on the data transfer types, each of these two states can be divided into 2 substates. They are sequential-single transfer state and burst transfer state. Taking the Idle state into account, the state machine of Core Unit 2 consists of 5 states totally. They are Idle state, sequential-single data read state, sequential-single data write state, burst data read state, burst data write state. Some of these states can go to another state after their execution is finished. When the data transfer is operating in a burst transfer mode, the number of the wait states and the location where wait states are inserted are important factors for the state machine design. Even for multiple wait states insertion, there are several different situations: continuous wait states insertion, incontinuous wait states insertion, and the combined situation of former two types. What's more, each burst consists of 4 data elements, which means there can be four possible options to insert wait states: before burst transfer, the first data elements completed, the second and the third data element, respectively. Therefore, the state machine should be analyze properly state to state.

3.3 Conclusions

In this analysis, the existing results failed to specify and implement the Core Unit correctly. A test principle was proposed to test the existing results. The Unit 1 was not well defined. For the Unit 2, the state machines are wrong. The wait states he inserted only last one clock cycle. But according to the concept of the wait states, more clock cycles can be consumed during one wait states. Based on these analyses, possible solutions are given.

Core Unit Design Implementation

4

This chapter focuses on the Core Unit specification and implementation. According to the requirement, Hatley and Pirbhai methodology is used in this chapter. Section 4.1 describes the Conceptual Architecture of the Core Unit. A structure of the DMA Controller based on the Core Unit Cells is proposed in this section. The function requirements are also presented. In Section 4.2, the Logical Architecture of each subunit is described. According to the analysis of Chapter 3, a new Address and Counter Management Unit is specified, and the state machines of Subunit 2 are designed to fulfill the function requirements. The Logical Architecture of the Core Unit is fully implemented using VHDL in the end of the design.

4.1 Core Unit Conceptual Architecture Specification

4.1.1 Core Unit Context Diagram

Figure 4.1 represents a block diagram of DMA Controller. In this figure, multiple Core Unit cells connect to different type of multiplexers. Each single Core Unit consists of a FIFO buffer with core control hardware for a single channel. Therefore, the number of the Core Unit cells equals to the number of DMA channels. After passing the channel multiplexer, the data will go into the FIFO buffer which is inside the Core Unit. After that, it will be then passed to the destination under the control of data transfer hardware. The path to the destination peripheral is arranged by a Peripheral Multiplexer. The data is finally transferred to destination under the control of Flow Control Unit whereby handshake signal is provided.

The Core Unit cell here can be treated as a DMA channel. Thus, it is easier to change the number of DMA channels by increasing or decreasing Core Unit cells. Therefore, when the number of AHB bus masters changes, the DMA Controller does not need to be redesigned. Using this implementation, the DMA Controller can adapt flexibly in different application environments. This design provides much convenience for further development in entertainment systems.

Linked List Unit is in the charge of linked list data transfer, which will be described later. Linked List Multiplexer is used to assign a DMA channel to the internal Core Unit cell in order to transfer linked list data in the next DMA cycle. Channel Scheduler makes sure that AHB bus master transfers data through the right DMA channel. According to the programmed channel priority, for each data transfer task, DMA Multiplexer is able to change from one DMA channel to another. When one DMA channel is selected, other DMA channels should be halted during its operation.

Since the data transfer is the most important functionality of the DMA Controller. Thus, the design is starting from Core Unit.

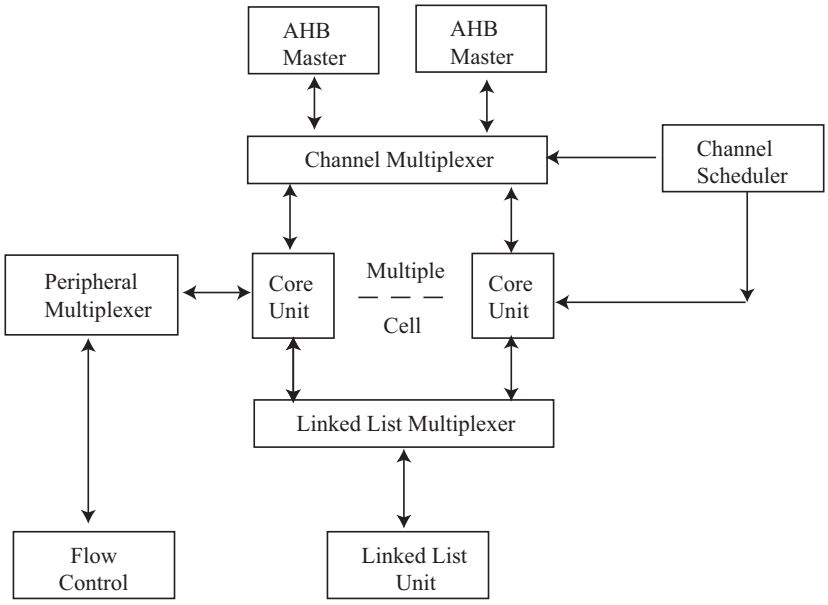

Figure 4.1: Core Unit Context Diagram

4.1.2 Core Unit Requirements and Constrains

4.1.2.1 Core Unit Requirements

1. Core Unit includes a corresponding Core Unit cell for each DMA channel.

The Core Unit cells are flexibly usable. During implementation, the system architect and software programmer can decide how many Core Unit cells should be placed inside the Core Unit.

2. Core Unit must support a sequence of DMA cycles for each clock cycle.

This DMA Controller is designed for the vehicle system. Continuous data transfer is very important to the transfer speed, as well as the precision of entertainment information, which may consist of different entertainment formats.

3. Core Unit must support both sequential-single and burst data transfer.

Burst transfer is a more efficient data transfer mode. Transferred data is normally generated into several bursts and a maximum of 3 sequential single DMA cycles.

4. Core Unit must support wait states insertion during both sequential-single and burst transfers.

Wait states are usually inserted into the AHB bus to balance different transfer speed between two devices. To minimize the number of wait states, data write process should be operated as soon as the read cycles are finished.

5. Core Unit must support Linked List transfers for scatter-gather DMA operation.

Linked list data transfer provides much convenience to collect data which is scattered all over the memory and gather these data elements into a consequent address area. Linked list transfer can support data blocks with different sizes, and the transfer is also operated without the interference from the microprocessor.

6. Core Unit must support data packing for transfers among different size of data elements.

Data type in the memory is usually different from the types in the microprocessor and other peripherals. Therefore, when data is transferred from one peripheral to the memory, for instance., data in the peripheral is firstly packed into the size of the data type in the memory and then transferred to the destination memory.

4.1.2.2 Core Unit Constrains

Memory address space is 32 bits. Data width is max 32 bits. Data transfer length is max 22 bits. Burst size is max 4 elements.

4.2 Logical Architecture Specification and Design

4.2.1 Process Model Hierarchy

Figure 4.2 below reflects the structure and hierarchy of the design. Each unit contains a Data Flow Diagram (DFD) and a Control Flow Diagram (CFD). Unit 0 is the top level specification, which describes the functions of the Core Unit. Normally, Unit 0 consists of several subunits. And the decomposition can also be seen as a function division. Each of the subunit takes responsible for parts of the functions of Unit 0.

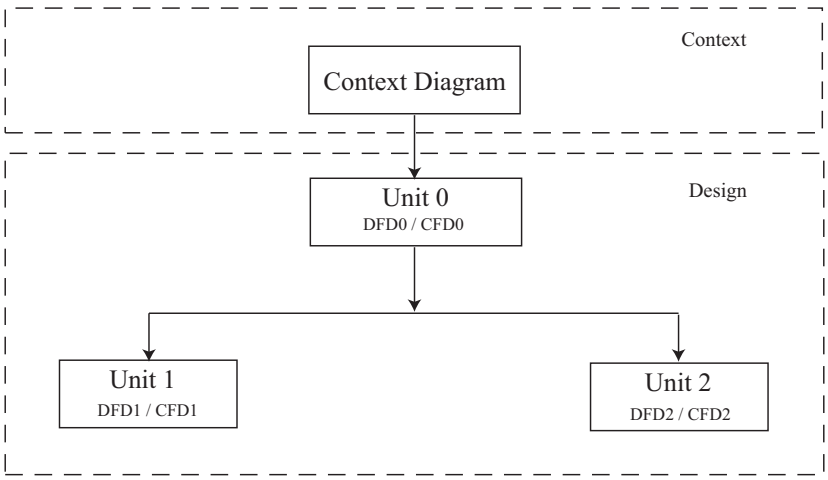

Figure 4.2: Core Unit Process Hierarchy

4.2.2 Top Level Logical Architecture Specification

According to the hierarchy diagram, the top level of the Core Unit, Unit 0, is divided into two functional processes, namely Unit1 and Unit 2. Unit 1 is a process to manage address and counters. It configures the address and read/write counters based on the information in the descriptor. Unit 2 is responsible for performing the data transfer from source address to destination address. Along with data transfer, Unit 2 triggers Unit 1 to update the values of address and counter on time. The Data Flow Diagram and Control Flow Diagram of Unit 0 are showed below, in Figure 4.3 and Figure 4.4, respectively.

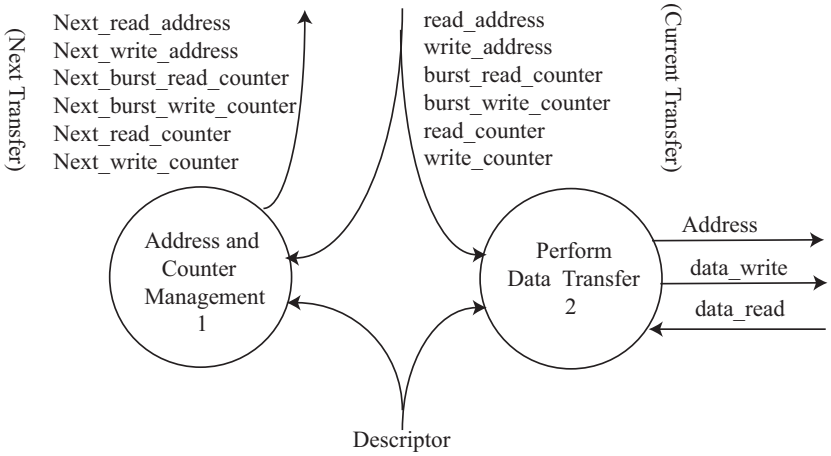

Figure 4.3: Core Unit 0 Data Flow Diagram

From the Data Flow Diagram, the descriptor provides the information to Unit 1, including source address, destination address and transfer length. According to these information, Unit 1 starts its operation by incrementing the read/write address and decrementing the value of read/write counter until the counter turns to 0.

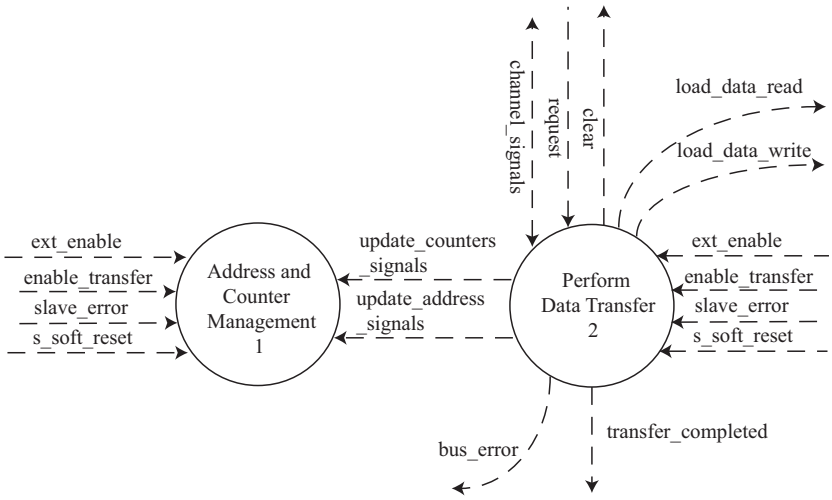

Figure 4.4: Core Unit 0 Control Flow Diagram

From control flow diagram, both of the two units can be triggered by external signals. Once the Unit 2 performs data transfer for one data item, it sends update signals of counters and addresses to Unit 1 as soon as possible. With the new address and counter information, Unit 2 is able to transfer the next data item.

4.2.3 Core Unit 1 Specification and Design

According to the problem analysis in Chapter 3, when inserting wait states into a burst transfer process, the counters and the burst transfer become incorrect. The existing results only consider the situation with one wait state insertion. However, the fact

is that there might be more than 1 wait state inserted into data transfer process to balance different speed between the source peripheral and memory, and vice versa.

To emphasize the counter management process, a new Flow Diagram is redrawn around these two units, which is shown in Figure 4.5:

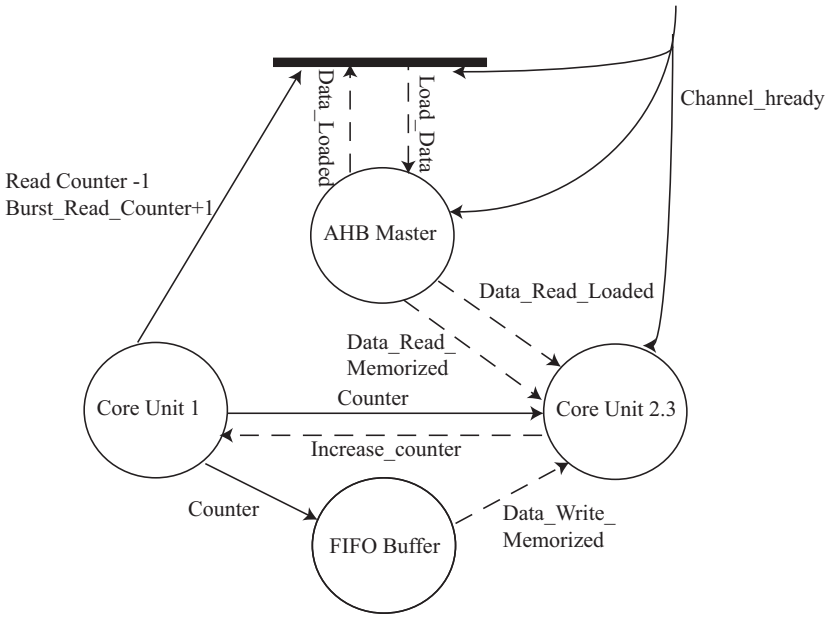

Figure 4.5: Core Unit 1 Analysis

I complement all the situations of wait states insertion and all the conditions needed to be involved as well. But later on, I find that it is hard to make it realized. As discussed above, the number of the wait states insertion depends on the situation of data transfer task to task, different tasks have different corresponding wait states insertion. Moreover, it is possible to finish and figure out all the situations, a large number of VHDL codes were required. Thus this would be a big load to the system design and also not cost-efficient. Therefore, new method needed to be found.

Since both of Core Unit 1 and Core Unit 2.3 are responsible for the counter management, and the main goal of counter management is to update the counter as soon as the data element is transferred, the Core Unit 2.3 can be merged into a Core Unit 1. In this way, the new Core Unit 1 is responsible for the counter and address management, and the new Core Unit 2, on the other hand, focuses on performing data transfer. In this way, some signals can be neglected. Based on their functions, the new Core Unit 1 is named as 'Address and Counter Management' and the new Core Unit 2 is named as 'Perform Data Transfer'.

By analysis the mechanism of counter increment and decrement operation, the value of the counters should be updated as soon as current data element is transferred. Using the data transfer control related signals to generate counter trigger signals can be taken into account. The related signals are 'load data read/write' and 'data read/write loaded'. Signal 'load data read/write' indicates when the read or write address are placed on the AHB bus. Taking the read process as an example on the rising edge of

the clock cycle, when the read or write address is detected to be updated, the 'load data read' signal becomes high. Otherwise, it will stay low. Take the And then on the next cycle, a data element is read from the source peripheral through the data bus. At this moment, the signal 'data read loaded' is raised to '1' until the data element is transferred. And then it returns to '0'. When there is a high 'load data read' signal, a data element is ready to be read and a read process is coming on the next cycle consequently. One data element is reading by the memory, while a 'data read loaded' signal is high at the same cycle. And on the next cycle, the counter will be updated. Meanwhile, another signal 'read address active' is needed to distinguish the read process out of the write process. The original signals 'data read/write loaded', 'channel hready', 'read/write counter' and 'burst read/write counter' can be neglected. Moreover, the 'data read/write memorized' signal is no longer useful, as well as the output signal, 'increase burst read/write counter', of original Core Unit 2.3. Thus, these two signals can be removed from the design. To sum up, the new counter management design is showed below:

Behavior	Conditions
burst read counter	data read loaded = '0' and load data read = '0' and read address active = '1'
burst read counter + 1	data read loaded = '1'
burst write counter	data write loaded = '0' and load data write = '0' and read address active = '0'
burst write counter + 1	data write loaded = '1'
read counter - 1	data read loaded = '1'
write counter - 1	data write loaded = '1'
read address + 1	increase read address = '1'
write address + 1	increase write address = '1'

Table 4.1 New Counter Management Design

From Table 4.1, once one data element is read or written, both the read or write counter and the burst read or write counter will be updated on the next cycle. This takes one clock cycle, comparing with the two clock cycles in the original design. Once one data element is loaded, on the first cycle, the 'increase burst read/write counter' signal will be generated and sent to the old Core Unit 1 to process the counter update on the next cycle. No matter how many wait states are inserted, the counter increment or decrement is able to perform correctly as long as the data transfer is correct.

4.2.4 Core Unit 2 Specification and Design

Core Unit 2 is 'Perform Data Transfer' unit. The function of this unit is to perform data transfer process from source address to destination address. When the destination address is ready to receive data, a trigger signal will be sent to this unit to start the data transfer. After that, Core Unit 2 starts the data transfer process till all the required data elements are completely transferred. Once the transfer task is finished, a 'transfer completed' signal will be triggered and the DMA Controller will halt and wait

for a new trigger signal to start a new transfer task. The configuration information for the data transfer task is preset in a descriptor instruction by the microprocessor and transferred to Core Unit 2 right before the execution. The Core Unit 2 operates along with the address and counter updates of Core Unit 1. As soon as one data element is transferred to its destination, a trigger signal will be sent to Core Unit 1 to update the next destination address and the value of the counters. When the write counter turns to 0, the whole execution of Core Unit 2 is finished.

According to the requirements of the DMA Controller, Core Unit 2 needs to support three different types of data transfer: sequential data transfer, burst data transfer and the combined data transfer of the former two types. Each transfer type must support wait states insertion. Core Unit 2 must also support Linked List data transfer.

4.2.4.1 Data Flow Diagram

According to the requirements and existing specification [2], Core Unit 2 can be specified into 5 processes. They are Core Unit 2.1 'Define Read', Core Unit 2.2 'Define Write', Core Unit 2.3 'FIFO Buffer', Core Unit 2.4 'Generate Address' and Core Unit 2.5 'Manage Linked List'. The Data Flow Diagram is shown in Figure 4.6.

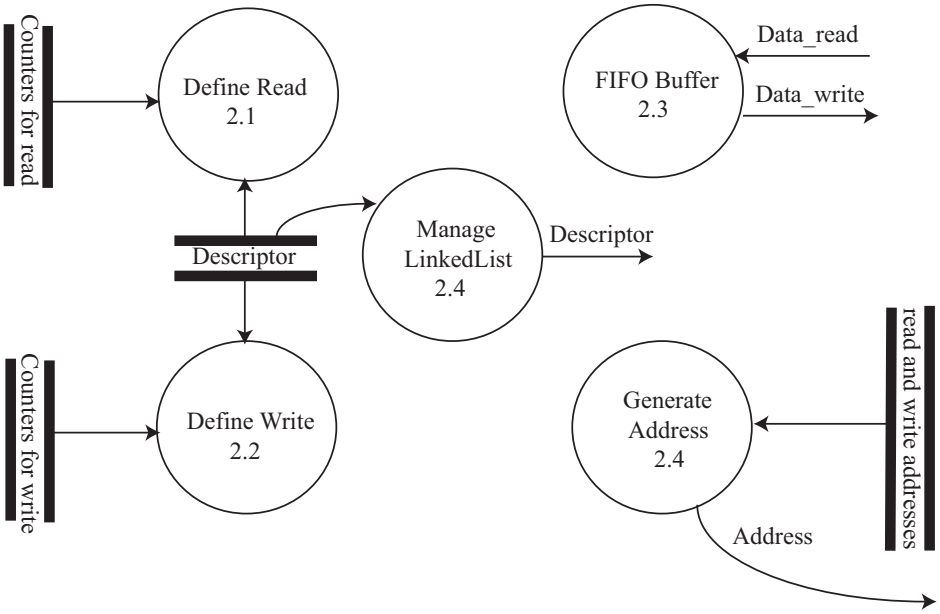

Figure 4.6: Core Unit 2 Data Flow Diagram

Process description

Core Unit 2.1 Define Read

Define Read process is responsible to decide whether it's a sequential-single read process or a burst read process based on the signals inside descriptor instructions. When signal 'burst read enabled' is '1' and the transfer length is larger than or equals to 4 data elements, which is the size of a burst of data, Core Unit 2 performs burst read process. And when 'Burst read enabled' is '0' and the transfer length is less than

4 data elements, a sequential single read process will be executed. After the type of read process is defined, a trigger signal will be sent to the state machine.

Core Unit 2.2 Define Write

Define Write process, like Define Read process, chooses the write process mode between a sequential-single and burst. The corresponding condition signals are 'burst write enabled'. A trigger signal is also generated when process is finished.

Core Unit 2.3 FIFO Buffer

FIFO Buffer is used to temporally store the read and write data. It's implemented with a four stages shift register [2].

Core Unit 2.4 Generate Address

Generate Address process will assign the read or write address signals to the address bus. The signal 'read address active' is used to distinguish read address and write address. When 'read address active' = '1', read address is assigned to the address bus. Otherwise, it's the write address.

Core Unit 2.5 Manage Linked List

Manage Linked List process is used to forward the descriptor to the Linked List Unit, and triggers the Linked List transfer through a 'Enable LinkedList' signal.

4.2.4.2 Control Flow Diagram

The Control Flow Diagram of Core Unit 2 describes the control signals among the specified processes. And it is shown in Figure 4.7.

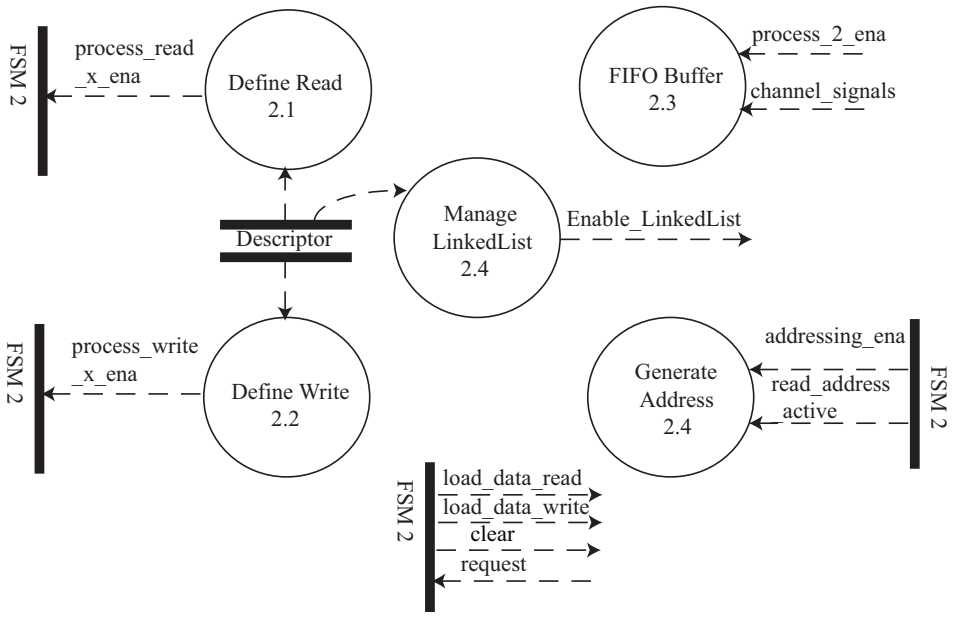

Figure 4.7: Core Unit 2 Control Flow Diagram

4.2.4.3 State Machine Design

There are two basic types of state machine: Mealy machine and Moore machine. In a Mealy machine, output signals of the state machine rely on both the input signals and the present state. Thus, if the value of input signals change, the value of output signals will be adjusted. In a Moore machine, output signals of the state machine only depend on the present state. For each state, there are unique output values irrespective of the inputs change. The selection of each machine will make the design different, and it depends on different applications. In practice, mixed machine is usually adopted as well.

As discussed in problem analysis, the state machine includes 5 states. They are Idle state, sequential-single data read state, sequential-single data write state, burst data read state and burst data write state. Some of the states are the following states of others. For example, sequential-single data write state comes right after sequential-single read state.

Idle State

The Idle State indicates that no data transfer is required. The IDLE transfer type is used when a bus master is granted the bus, but does not wish to perform a data transfer. DMA Controller has been holding the authority of the AHB bus, but there is not any data transfer request. So Core Unit 2 is simply waiting for the trigger signal to start data transfer task. The state machine can be seen in Figure 4.8.

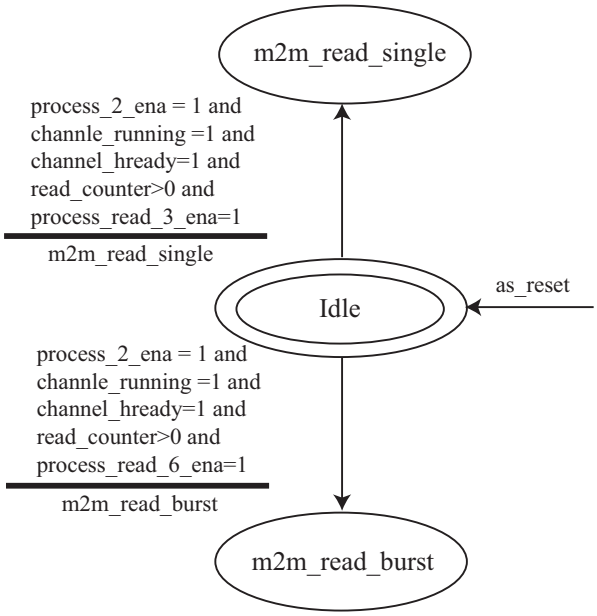

Figure 4.8: Core Unit 2 FSM Idle State

sequential-single data read state

The sequential-single data read state is used to read the single data element from the source memory to the FIFO buffer inside DMA Controller. The state machine is a Mealy machine. If read counter changes to different values, the output signal will change correspondingly. As a consequence, the state can transit to the sequential-single

write state or Idle state.

Owing to different data transfer speed between the source memory and the DMA Controller, wait states can be inserted into the read process. For each sequential single read process, DMA Controller takes one clock cycle to transfer one data element. Therefore, wait states can only be inserted before the transfer. Wait states are triggered by signal 'channel ready'. The signal keeps high when there's no wait state. As soon as the signal becomes low, wait states triggered. Different source device demands different number of wait states. Figure 4.9 shows the case with 1 wait state insertion. Indicated by the figure, all the processes are halted during the wait state. However, even it's in the read state, the signal 'load data read' still keeps low in wait period. This is because the signal is used to trigger the read process. Once the signal becomes 1, on the next cycle, data will be read immediately. During the wait state, the read process should be halted, and therefore, the signal becomes low.

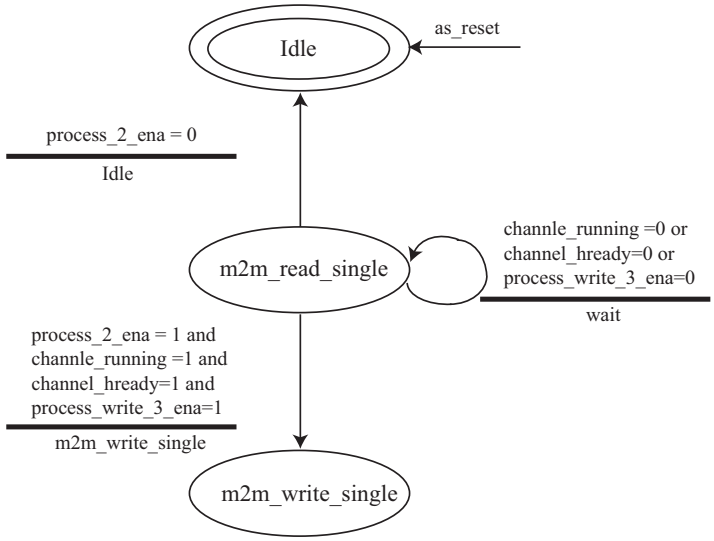

Figure 4.9: Core Unit 2 FSM Sequential-Single Read State

The following state of the sequential data read state is sequential single write state. To make this transition, the single write trigger signal 'process write 3 ena' should be high. The condition to activate the signal is 'burst write enabled' signal, 'write counter' and 'burst write counter'. The 'burst write enabled' is used to trigger the burst write process. Therefore, it is set to 0 as default. As mentioned before, both Idle and burst data write state can transit to sequential single read state. The former case represents the start of data transfer task. The value of write counter is less than 4. The latter case is the last three data elements in transfer task. Due to different wait states insertion cases of the burst write process, the 'process write 3 ena' should be triggered 2 data elements before the sequential single write process. To satisfy this, one of the conditions is that the 'burst write counter' equals to 2 corresponding with value 5 of 'write counter'. Moreover, 'burst write counter' is 3 and 'write counter' is 4 at the same time. Once the 'read counter' reduce to 1, the read address should not increase anymore. Thus, the output signal 'increase read address' becomes 0. When the value of 'read counter' is larger than 1, the signal is 1 in order to continue the read

process. For both situations, the 'load data read' signal is set to 1. The data element can be read during the data phase on the next clock cycle. The sequential single read state can also transit to Idle state. However, it only occurs when Core Unit 2 fails to be enabled.

sequential-single data write state

The sequential-single write state takes responsible for writing the data element to the destination memory. This state is a Mealy machine. Based on different conditions, such as read and write counter, the state can transit to sequential-single read state or Idle state. When the state transits to the sequential-single read state, DMA Controller starts to read the following data element. If the next state is Idle, current data transfer task will be finished after the data element is written. And DMA Controller will send an interrupt signal to the microprocessor to halt the DMA operation. The state machine can be seen in Figure 4.10.

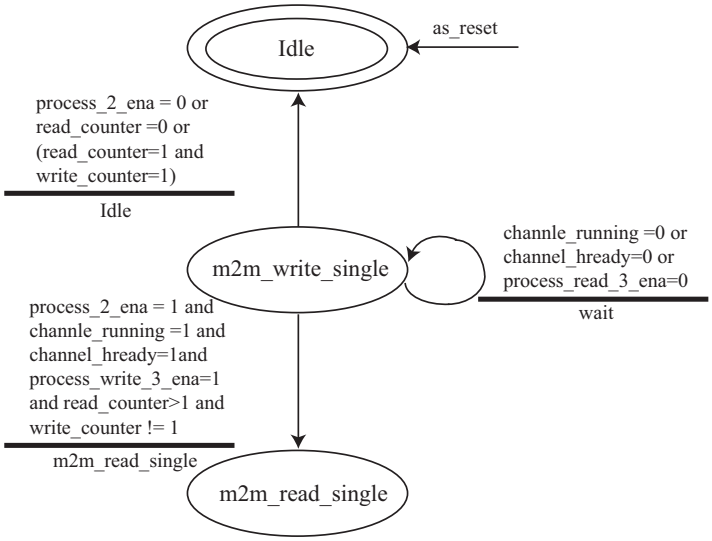

Figure 4.10: Core Unit 2 FSM Sequential-Single Write State

In the sequential single data transfer process, the sequential-single write state follows the sequential single read process, which is finished when the data element is read into the FIFO buffer. Under Sequential single write state, the data read control signal should be hold to 0. Thus, during the Sequential single write state, the read address will not be increase. New data element can't be read until another read process happens. According to AMBA AHB operation, the address phase of the sequential single write process occurs during the write state, and the data phase occurs on the following state, which is sequential-single read state or Idle state. During the address phase, the destination address located in the memory is placed on the address bus. A data write trigger signal 'load data write' and make it high to load the data from FIFO buffer. Then, in the following data phase, the data element is transferred to the destination address. The signal 'data write loaded' becomes s high to track this process. When 'data write loaded' returns to low, the write process is finished. On the next cycle, the write counter subtracts 1. The burst write counter, in the sequential single write state, counters either 1 or 0. Once the data element is written to memory, the burst write

counter becomes 1. When the state is sequential single read, the burst write counter is 0.

The sequential single write process performs data write process to the destination address. The data transfer speed may be different between DMA Controller and destination memory. Therefore, wait states can be inserted into the process. The sequential single write process takes one clock cycle because of transferring one data element. Thus, the wait state can only take place before the transfer starts.

As we can see, all the signals are halted during wait states. As the data phase of the sequential single read state, the data element is read during sequential single write state. At the rising clock of the write state, the wait state trigger signal 'channel hready' still keeps high before wait states occur. As a consequence, DMA Controller starts to read the data element and the signal 'data read loaded' becomes high. But right after 1 clock cycle, 'data read loaded' turns to low. This means DMA Controller has already received one data element.

The DMA Controller is designed to perform both the burst and sequential single transfer. At the same time, the burst transfer comes before the sequential single transfer. Therefore, when DMA Controller stays in the sequential single state, either only three data elements existing in the data transfer task, or it's nearly the end of the data transfer. The latter case means the number of the remaining data elements is maximally only three data elements left and is not sufficient enough to create a data burst. Therefore, sequential single write state can only transit to sequential single read state or Idle state.

In order to transit to sequential single read state, the trigger signal 'process read 3 ena' must become high. The conditions are 'read counter' and 'burst read enabled' signal which is the configuration signal in descriptor instruction. As we discussed above, the maximum number of the remaining data elements is three, so the 'read counter' should less than 4. Since the expecting state is sequential single read, the burst read transfer should not be activated. As a consequence, signal 'burst read enabled' is low.

Based on the activated 'process read 3 ena' signal, if the 'read counter' equals to 0 or both the read and write counter is 1, the following state will be Idle. The former condition means all the data elements have been read by DMA Controller. Thus, after the last write process. the data transfer task can be finished. The latter condition represents there is still one data element left. Since the current state is sequential single write, which means DMA Controller is processing the write operation for the last data element. And the following state will certainly be Idle state. Otherwise, when the 'process read 3 ena' still keeps high, the state will transits to sequential single read state.

Burst data read state

Burst data read state describes the burst data read process. The DMA Controller reads a burst of data from the source memory to FIFO buffer. A burst of data consists of four data elements. In this state, DMA Controller reads the data elements of the burst continuously. The burst of data is firstly stored in the FIFO buffer. Then, after all the data elements in the burst are read, the DMA Controller transits to burst data write state. The burst data read state is a Mealy machine. Similar to the sequential single read state, the following state of burst data read state is burst data write state

and Idle state. The latter case only occurs when Core Unit 2 is not enabled. The state machine can be seen in Figure 4.11.

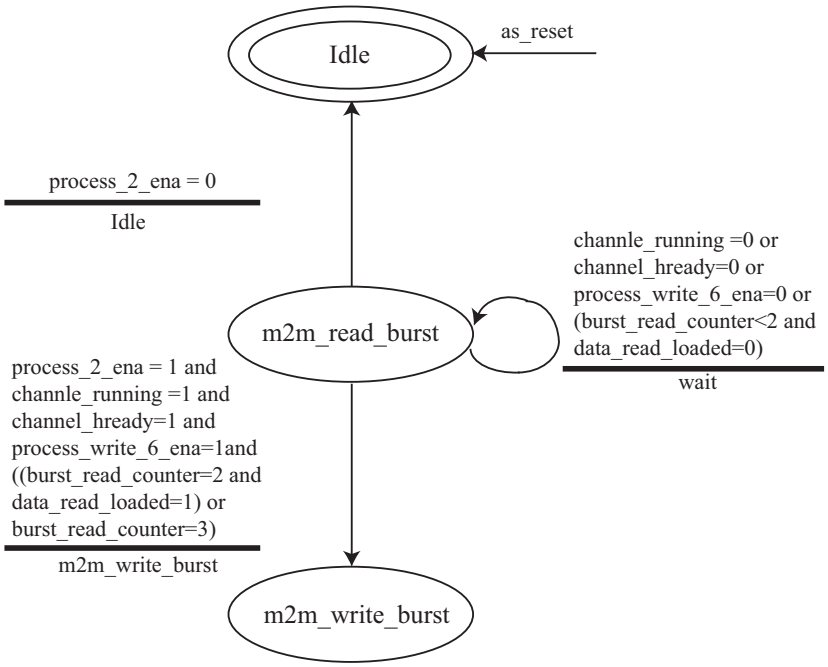

Figure 4.11: Core Unit 2 FSM Burst Read State

Two states can transit to burst data read state, which are Idle state and burst data write state. Transiting from Idle state means the beginning of data transfer task. To trigger burst read process, the signal 'process read 6 ena' must become high. The condition signals are 'burst read enabled', 'burst write enabled' and 'read counter'. Because of 4 data elements in a burst, the 'read counter' must larger than or equal to 4. Certainly, the 'burst read enabled' must be high, as well as 'burst write enabled' signal. This is to make sure all the required data elements can be read throughout the burst transfer process.

During the burst read process, the signal 'read address active' becomes high at the rising edge of the clock cycle. The read address is increased and placed onto the address bus continuously till all the four data elements is read. The write address increment needs to be halted. And the data write process can't be triggered as well. Therefore, both 'increase write address' and 'load data write' signals must set to 0. On the contrary, the 'increase read address' and 'load data read' signals should last high consequently for 4 clock cycles. Due to operation above, the data elements can be read continuously. As a consequence, the 'data read loaded' signal keeps high synchronously with the data read process. When the signal returns to low, the burst read process is finished.

Wait states can be inserted the burst read process. There are totally four data elements in a burst. Therefore, it's possible to insert wait states before the read process, or between every two data elements. The burst read state is able to transit to burst data write state or Idle state. During the burst data write state, the burst of data,

which is now stored in FIFO buffer, is written from DMA Controller to the destination memory. The trigger signal to this transition is 'process write 6 ena' signal. The input signals are 'burst read counter' and 'data read loaded'. The data elements are read during the clock cycles when 'channel hready' is high. The data phase occurs one clock cycle later than the address phase. And the first clock cycle of a state is taken by the address phase. When there's no wait states insertion, the data phase of the last data element transfer occurs at the first clock cycle of the following state. If the current transfer is burst transfer mode, the corresponding value of burst counter at the rising edge of the following state should be 3. Thus, the input 'burst read counter' should equal to 3. If the wait states are inserted into the first or second data element transfer, the corresponding burst counter may be 2. This delay comes from one of the features of the wait state. This characteristic is to extend the transfer cycle of one data element. In this case, the input 'burst read counter' should be 2. Apparently, the second data element must also be loaded at the moment. Thus, the input 'data read loaded' must be high either. Then the burst data read state can transit to burst data write state.

Burst data write state

During burst data write state, the DMA Controller writes the data burst from FIFO buffer to the destination memory. Similar to the burst data read state, all four data elements are written to destination address continuously. To trigger this state, the signal 'process write 6 ena' must be high. Therefore, the 'burst write enabled' signal must be 1 and 'write counter' should be larger than 4. The state machine can be seen in Figure 4.12.

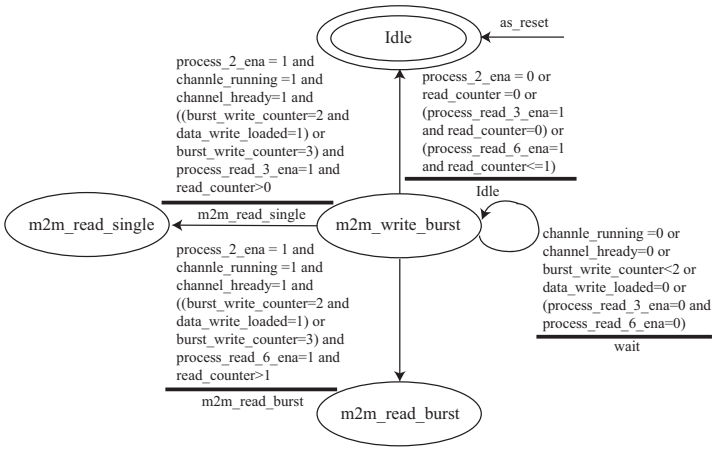

Figure 4.12: Core Unit 2 FSM Burst Write State

The burst data write state is also a Mealy state. The output signals are 'increase write address' and 'load data write'. Their values are influenced when input signals change. The input signals includes the 'burst write counter' and 'data write loaded'. There are three possible following states for the burst write state. They are burst data read state, sequential single read state, Idle state. When the burst transfer continues, the state is going to burst read process. If the number of remaining data elements is less than 4, the burst transfer will transit to sequential single transfer. Thus, the state is going to sequential single read state. If the number of required data elements

happens to be multiples of 4, the transfer will be finished right after the last burst writes process. Then, the state will transit to Idle state. Two input signals decide the transition. They are 'burst write counter' and 'data write loaded'. The analysis of the transition is the same as the burst data read process. Therefore, either the 'burst write counter' equals to 2 and 'data write loaded' is 1, or 'burst write counter' is 3.

4.3 Conclusion

The Core Unit of the YAD-C DMA Controller is fully specified and implemented in this chapter. The address and counter processes of the new Subunit 1 'Address and Counter Management' are fully controlled by the data read or write process. The new specification makes sure that the address and counters can be updated following the data transfer process closely. As long as the implementation of the data transfer functions is correct, the address and counters can be updated accurately. The Subunit 2 is required to realize the both sequential-single transfer and burst transfer with wait states insertion correctly. The data transfer functions, which are realized by the Subunit2 'Perform data transfer', are controlled by the state machines. There are five states totally: Idle state, burst read state, burst write state, sequential-single read state and sequential-single write state. The transforms among these states are fully analyzed, especially when the wait states are inserted. Therefore, the state machines are able to fulfill all the required data transfer functions. The Core Unit is implemented using VHDL after the specification.

Linked List Design Implementation

5

This chapter describes the specification of the Linked List Unit. Section 5.1 introduces the concept of Linked List transfer and proposes the Conceptual Architecture. In this section, the functional relation between the Linked List Unit and the Core Unit is also discussed, and a set of requirements of the Linked List Unit is presented in the end. In section 5.2, the Logical Architecture and its corresponding specification are described, which is a base for the future implementation.

5.1 Conceptual Architecture Specification and Design

5.1.1 Introduction of Linked List Transfer

A linked list is a list of so-called "descriptors" that consists of the source address of data blocks on several memory locations, destination address of each data block, transfer length, some configuration options and the pointer to the next descriptor in the list.

5.1.1.1 Why Linked List Transfer Is Adopted

In many cases, the data elements being transferred to system memory are dispersed into locations throughout the memory address space. Scatter-gather is a shortcut method of input and output operation which automatically gathers/scatters data blocks from/to non-continuous multiple memory locations to/from one continuous area in a single procedure. Scatter-gather requires the microprocessor programs a linked list of descriptors whereby the data elements can be scattered or gathered.

5.1.1.2 Different Types Of Linked List Transfer

There are two types of Linked List: Static Linked List and Dynamic Linked List.

Static Linked List

The descriptors of a Static Linked List are programmed in memory before the transfer starts. During the transfer, the descriptors can't be modified. When all DMA transfers are finished, the microprocessor is notified. The process is showed in Figure 5.1.

Dynamic Linked List

The purpose of the Dynamic Linked List is to move only payload out of a frame, which consists of a header, payload, tailer, whereby the length of the payload is variable. This is used when the data is transferred through the network in the form of packet. The descriptor of the Dynamic Linked List is read by the DMA Controller and it points to the header of the next frame when one frame is transferred. The length of

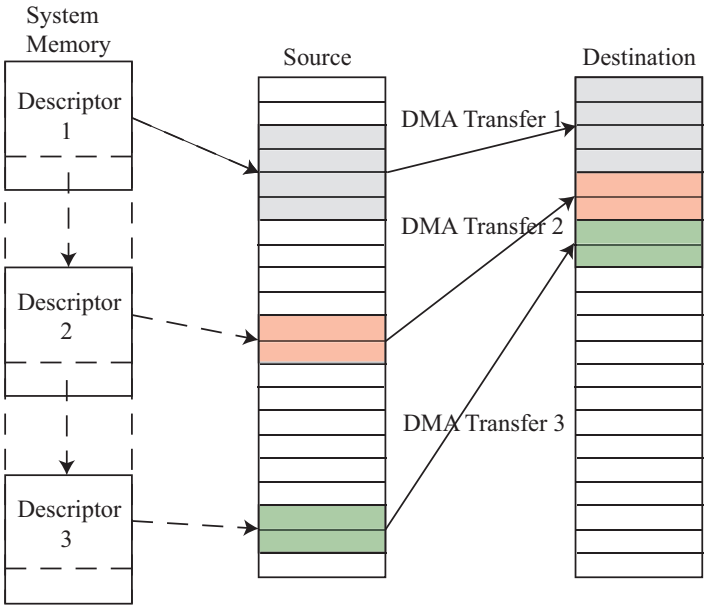

Figure 5.1: Static Linked List Process [1]

the payload is extracted from the header. Thereafter, from a fixed offset of the source address, the payload is read. The process is shown in Figure 5.2.

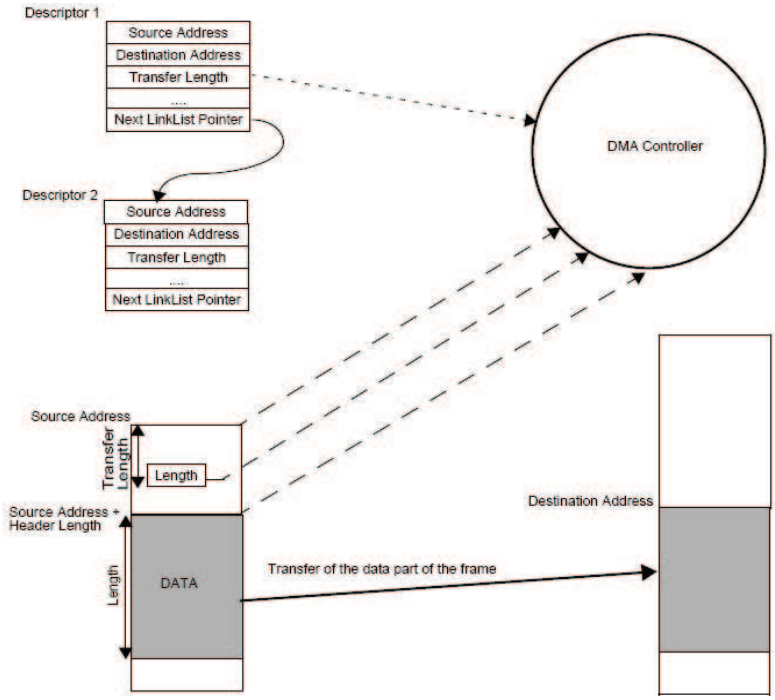

Figure 5.2: Dynamic Linked List Process

The descriptor of Static Linked List is the same as the descriptor of normal transfer. However, the descriptor of Dynamic Linked List is different. Table 5.1 shows the

structure of Dynamic Linked List descriptor and necessary description.

Instruction	Explanation
Source address	The source address of frame header
Destination address	The address of destination memory
Transfer Length	The length of frame header is offset of data field of the packet.
Dyn LinkedList ena	Start Dynamic mode when "'1"', Static mode when "'0"'
Transfer Type	The format of the data element: word, byte, bit
Descriptor Length	The amount of descriptors in the Linked List
Transfer Configuration	Define the destination peripheral, burst read/write enable
Next Descriptor Pointer	The source address of next descriptor

Table 5.1 Linked List Descriptor Structure

5.1.1.3 Advantages Of Linked List

Linked List transfer is able to realize a continuous DMA transfer in several incontinuous memory areas. Compared with normal transfer, there are two main advantages. Firstly, Linked List transfer makes the DMA Controller more intelligent, and increases the data transfer speed of DMA Controller. Secondly, the descriptors can be stored in different areas in the memory. Therefore, the number of descriptor items is not limited by its item length. Consider the normal transfer situations for some incontinuous DMA transfers. Once one data packet transfer is finished, the DMA Controller must sent a interrupt signal to the microprocessor the stop DMA process. Then in order to transfer the next data packet, DMA Controller has to be initialized again. This interrupt operation is a software process. Thus, if the executing time of the interrupt process is slow, whereas the second data packet requires for a quick response, errors may happen. On the other hand, during Linked List transfer, the DMA Controller reads the descriptor from the memory. It is a hardware process. This avoids the unnecessary delay of the software execution. As a consequence, the data transfer speed is increased. Software execution also occurs in the Linked List process. It is used to set the initialization information of the DMA Controller. This information can be set at the very beginning of the transfer. Therefore, such software execution also avoids to cause the possible errors.

5.1.2 Conceptual Architecture Specification and Design

5.1.2.1 Design Consideration

The Core Unit is responsible for the data transfer process according to the programmed descriptor instructions. Therefore, an unit should be designed to program the Linked List descriptor and sent it to the Core Unit before data transfer process. This unit is Linked List Unit. The Linked List transfer can be separated into Static Linked List transfer and Dynamic Linked List transfer. The procedures are different between each transfer mode.

Once a Linked List transfer is triggered, the Core Unit enables the Linked List process and transfers the descriptor to the Linked List Unit. The Linked List Unit

starts to read the descriptor from the source address. After reading, the Linked List Unit programs the descriptor and creates a new Linked List descriptor, which contains the necessary instructions to run a Linked List transfer. The Core Unit then reads the Linked List descriptor and performs data transfer. For each transfer period, one data packet is transferred. When transfer is finished, the above procedure is repeated till all the data packets are transferred. Then the Linked List transfer is complete. The whole process requires Register Bank to record the transfer status and control signals. According to the procedure, context Diagrams can be developed. The context diagrams can be seen in Figure 5.3 and Figure 5.4.

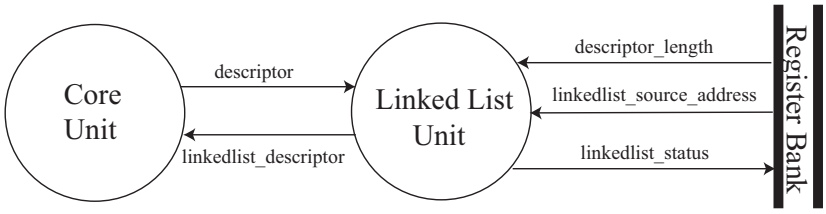

Figure 5.3: Linked List Context DFD

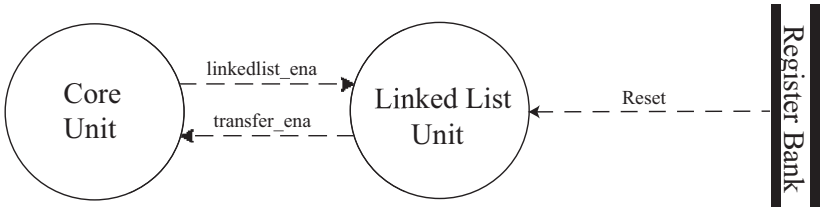

Figure 5.4: Linked List Context CFD

The Data Flow Diagram consists of three processes, Each of the processes has the following functions:

Core Unit

Core Unit Defines the Linked List transfer and perform Linked List data transfer. The descriptor contains a 'Linked List Transfer' signal. When 'Linked List Transfer' is true, the Core Unit must perform Linked List transfer. And the descriptor is sent to the Linked List Unit. Core Unit is able to trigger Lined List Unit by sending a 'linkedlist ena' control signal. After receiving the Linked List descriptor from the Linked List, the Core Unit starts to transfer data packet.

Linked List Unit

Linked List Unit is responsible to configure the descriptor into Linked List descriptor and forwards it to Core Unit. The process also reports the status of the Linked List transfer to Register Bank. When the status is '1', Linked List transfer continues. When it's '0', the transfer is finished. When the Linked List descriptor is ready, Linked List Unit must enable Core Unit to perform the data transfer.

Register Bank

Register Bank is a store that holds the configuration information of Linked List Unit. 'descriptor length' records the number of descriptors in a Linked List transfer

task. ‘linkedlist source address’ is the start address of the first Linked List descriptor. The Linked List Unit can also reset by the "‘reset"’ control from the Register Bank.

5.1.2.2 Requirement of Linked List Unit

Linked List Unit must both support Static Linked List mode and Dynamic Linked List mode. For Dynamic Linked List mode, the length of data field must be filtered from the frame header. And because the information hidden in the header is on bit level. Thus, the process must be done also on bit level. When the transfer is complete, the next descriptor pointer must point to 0.

5.2 Logical Architecture Specification and Design

5.2.1 Process Model Hierarchy

Figure 5.5 shows the hierarchy of the logical architecture of Linked List Unit. The logical architecture consists of 4 function units. When the function unit is complex, it can be further divided into subunits. Each unit is specified into corresponding Data Flow Diagram (DFD) and Control Flow Diagram (CFD).

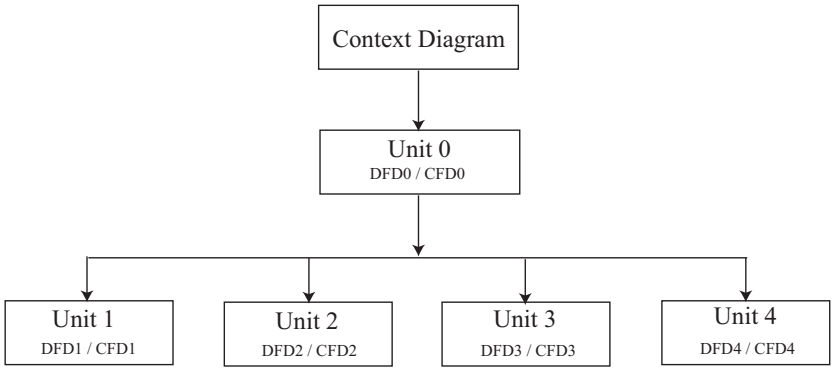

Figure 5.5: Linked List Process Hierarchy

5.2.2 Logical Architecture Specification

5.2.2.1 Data Flow Diagram and Control Flow Diagram

When a Linked List transfer is triggered, Linked List Unit must read the descriptor from the microprocessor via Core Unit. Then the Unit judges the Linked List transfer type to see whether the transfer is Dynamic Linked List mode or Static Linked List mode. A Dynamic mode means a data packet which is waiting to be transferred. So Linked List Unit should read the frame header and filter out the source address and transfer length of the data field of the packet. After the information is ready, the unit creates a Linked List descriptor. Then Linked List Unit finds the source address of next descriptor and starts to read. Linked List Unit also need to trigger the Core Unit to perform the Linked List data transfer.

According to the requirements and the function description above, the Data Flow Diagram can be developed. Figure 5.6 and Figure 5.7 shows the Top Level DFD and CFD.

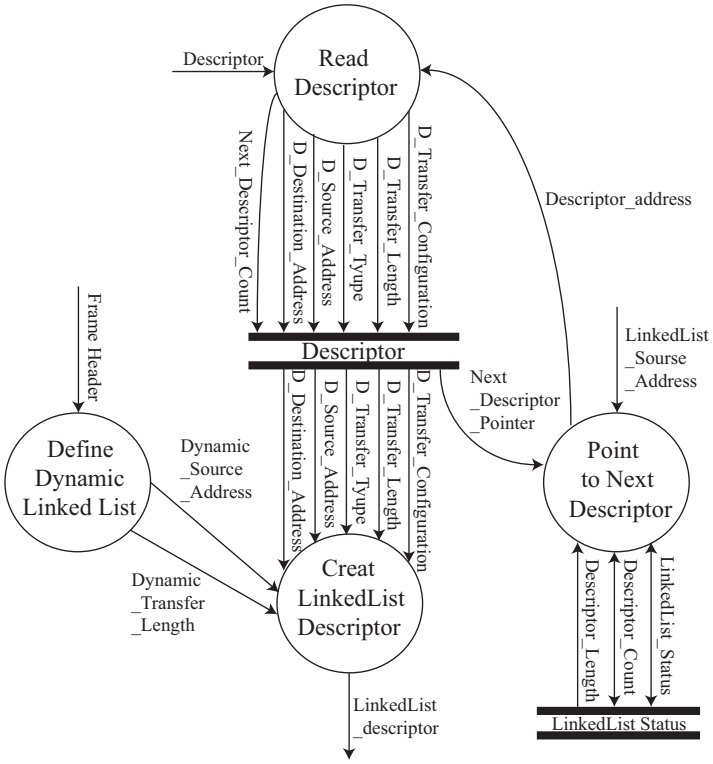

Figure 5.6: Linked List Unit 0 DFD

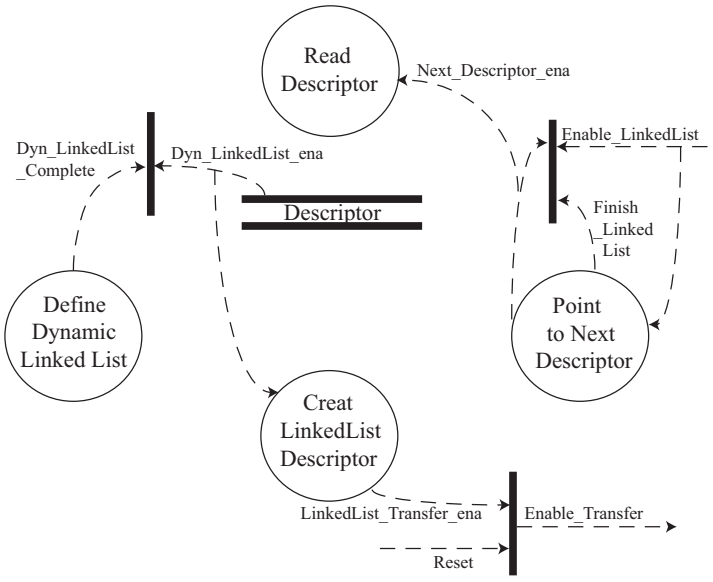

Figure 5.7: Linked List Unit 0 CFD

Lined List Unit consists of four processes and two store. They are Unit 1 'Read Descriptor', Unit 2 'Define Dynamic Linked List', Unit 3 'Create Linked List Descriptor', Unit 4 'Point to Next Descriptor' and Store 'Descriptor', 'Linked List Status'. Each component can be seen as a logical subunit. The top level DFD indicates the among each functional subunit. The input and output data is shown in Table 5.2.

Input	Output
descriptor	linkedlist descriptor
linkedlist source address	
frame header	

Table 5.1: Linked List Unit Top Level Input and Output

'Descriptor' is the original data descriptor. The instructions inside linked list descriptor are different from the normal descriptor. The descriptor contains 'source address', 'destination address', 'dyn linkedlist ena', 'transfer type', 'transfer length', 'transfer type', 'transfer configuration' and 'next descriptor pointer'. For Static Linked List mode, the information inside linked list descriptor is the same as the original descriptor. For Dynamic Linked List mode, the 'source address' is the source address of frame header, the 'transfer length' is also the length of frame header.

'linkedlist source address' is the source address of the whole linked list, and also the first descriptor. Linked List Unit uses this address to locate the position of the linked list in memory.

'frame header' is the header of the packet which is waiting for transfer. A data packet consists of header, data field and tailer. And the data field is the place that stores the required data elements. The length of the data field is stored in the header in bit level. In order to transfer the required data, the length information has to be extracted from the header.

The output data 'linkedlist descriptor' contains the programmed information that can be used by Core Unit for the data transfer. After programming, the 'linkedlist descriptor' is as same as the normal transfer descriptor. Thus, the data elements can be transferred as normal transfer process for each descriptor.

The top level CFD receives one input control signal "'enable linkedlist"' and generates one output control signal 'enable transfer'. The control signal 'enable linkedlist' comes from Core Unit. It is used to start the linked list process. After the 'linkedlist descriptor' is programmed, control signal 'enable transfer' is sent to Core Unit to trigger the data transfer process.

5.2.2.2 Linked List Subunits description

Linked List Subunit 1: Read Descriptor

This subunit is used to read the descriptor instructions from Core Unit. Since the function of the subunit is not complex, there is no need for further function division. The DFD and CFD of this subunit can be seen in Figure 5.8 and Figure 5.9.

This unit receives instructions one by one, and forwards to the descriptor store word by word. After the first descriptor in the Linked List is programmed, the unit starts

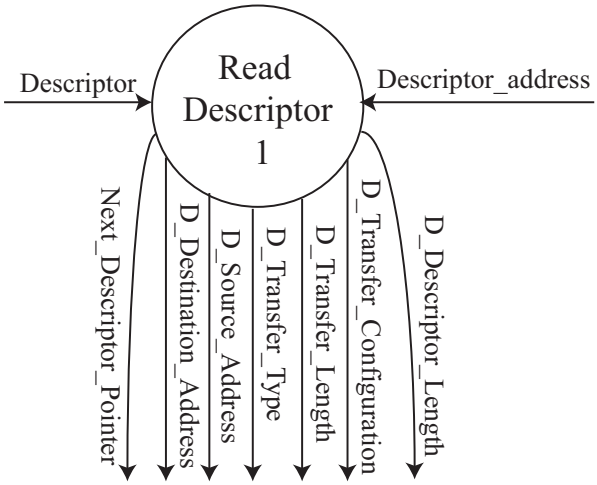

Figure 5.8: Linked List Unit 1 DFD

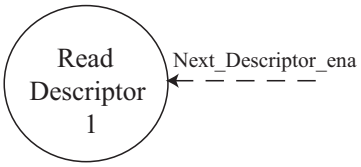

Figure 5.9: Linked List Unit 1 CFD

to read the second descriptor according to the 'descriptor address'. This process is triggered by a 'next descriptor ena' control signal.

In order to distinguish the descriptor instructions from the 'linkedlist descriptor', the output instructions are all added a title 'D', which means the original 'descriptor' from Core Unit. For example, the output destination address is 'd destination address'. The Process Specification (PSPEC) is designed.

PSPEC 1

---

issue D SOURCE ADDRESS = DESCRIPTOR ADDRESS
issue D DESTINATION ADDRESS = DESTINATION ADDRESS
issue D TRANSFER TYPE = TRANSFER TYPE
issue D TRANSFER LENGTH = TRANSFER LENGTH
issue D NEXT DESCRIPTOR POINTER = NEXT DESCRIPTOR POINTER
issue D TRANSFER CONFIGURATION = TRANSFER CONFIGURATION

Linked List Process 2: Define Dynamic Linked List

The unit 'Define Dynamic Linked List' is responsible to fix the start address and length of the data field from the input 'frame header' when Dynamic Linked List mode occurs. The Dynamic Linked List is triggered by descriptor instruction 'dyn linkedlist ena'. The example packet is showed in Figure 5.10.

According to the function description, this process can be further divided into three

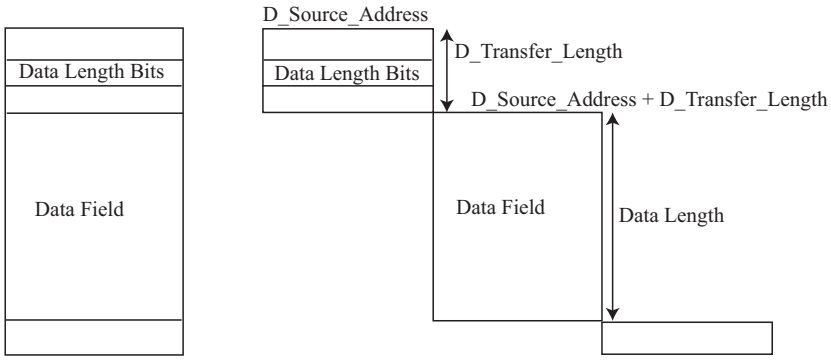

Figure 5.10: Linked List Unit 2 packet example

subunits. They are Subunit 2.1 'Header Filter', Subunit 2.2 'Transfer Length Buffer', Subunit 2.3 'Define Data Source'. The corresponding DFD and CFD can be seen below. After the process, the Dynamic Linked List source address and Dynamic Transfer Length will be sent to create the Linked List descriptor. The corresponding DFD and CFD are shown in Figure 5.11 and Figure 5.12.

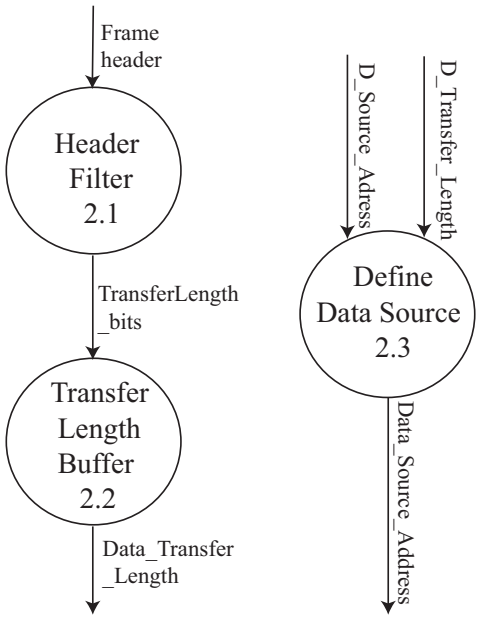

Figure 5.11: Linked List Unit 2 DFD

The Process Specification can be developed as below:
PSPEC 2

---

if Dyn LinkedList ena = 1 then
issue DATA SOURCE ADDRESS = D SOURCE ADDRESS + D TRANSFER LENGTH
issue DATA TRANSFER LENGTH is TRANSFERLENGTH BITS

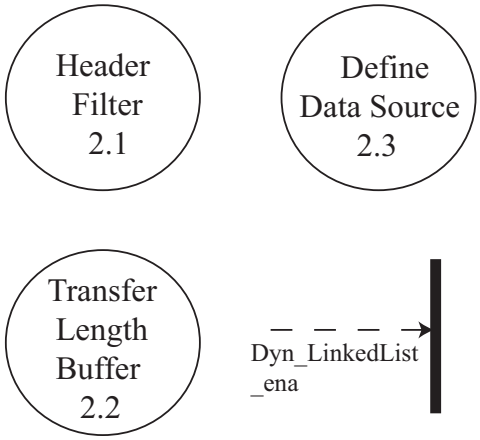

Figure 5.12: Linked List Unit 2 CFD

Subunit 2.1 Header Filter

'Header Filter' is used to filter the length of the data field from frame header. The filter can be designed as a shift circuit to shift the data length bits to 'Transfer Length Buffer'. The input and output signals can be seen in Table 5.3.

Input	Output
Frame Header	TransferLength Bits

Table 5.2: Linked List Unit 2.1 Input and Output

Subunit 2.2 'Transfer Length Buffer'

'Transfer Length Buffer' is used to store the data field length bits which is collected from the header. When the data length information is fully extracted, the subunit sends the data field transfer length 'data transfer length' to Linked List Unit 3 to create Linked List descriptor. The input and output signals can be seen in Table 5.4.

Input	Output
TransferLength Bits	Data Transfer Length

Table 5.3: Linked List Unit 2.2 Input and Output

Subunit 2.3 'Define Data Source'

This subunit is used to find the start address of the data field of the packet. When the Dynamic Linked List process is enabled, the subunit receives the source address and transfers length from the descriptor. The source address here is the start address of frame header, and the transfer length here is the header length. Based on these two instructions, the start address of data field can be calculated, which can be seen in PSPEC 2. The input and output signals can be seen in Table 5.5.

Linked List Process 3: Create LinkedList Descriptor

The function of the 'Create LinkedList Descriptor' process is to create the Linked List descriptor. When Dynamic Linked List occurs, the transfer length should be the

Input	Output
D Source Address	Data Source Address
D Transfer Length	

Table 5.4: Linked List Unit 2.3 Input and Output

length of the data field of the packet. And the source address must be also be the start address of the data field. Otherwise, the data transfer will be wrong. When it is Static Linked List, the source address and transfer length keeps the same as its original setting. According to this analysis, the process can be divided into three subprocesses. They are Subunit 3.1 'Define Transfer Length', Subunit 3.2 'Define Source Address'. The corresponding DFD and CFD can be seen in Figure 5.13 and Figure 5.14.

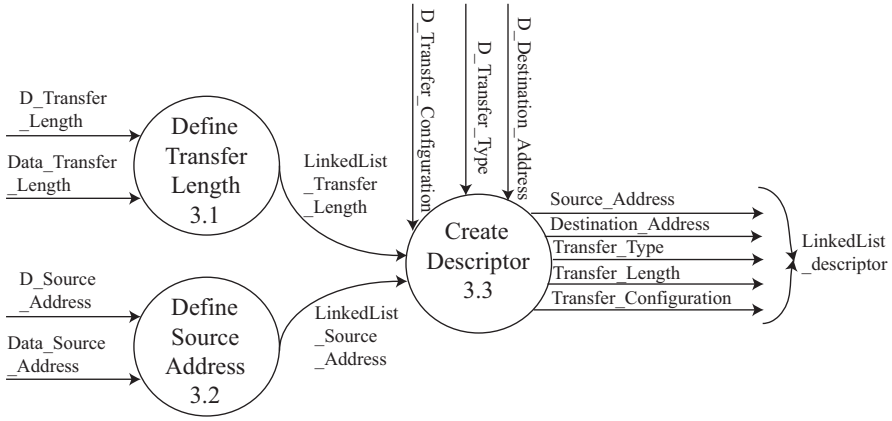

Figure 5.13: Linked List Unit 3 DFD

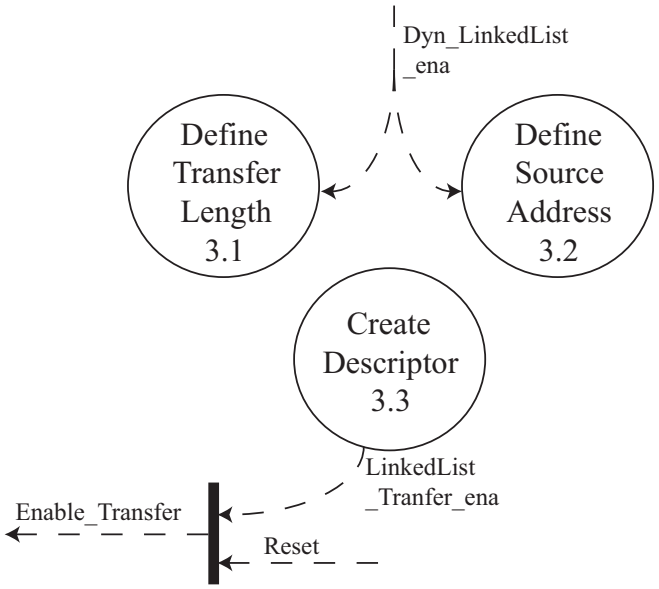

Figure 5.14: Linked List Unit 3 CFD

When Dynamic Linked List mode occurs, the process has to operate after the 'Data Transfer Length' and 'Data Source Address' are ready. If it is in Static Linked List mode, this unit can process as soon as the Linked List is enabled. The input and output can be seen in Table 5.6.

Input	Output
D Source Address	Source Address
Data Source Address	
D Destination Address	Destination Address
D Transfer Type	Transfer Type
D Transfer Length	Transfer Length
Data Transfer Length	
D Transfer Configuration	Transfer Configuration

Table 5.5: Linked List Unit 3 Input and Output

The Process Specification can also be developed according to the input and output signals.

PSPEC 3

---

if DYN LINKEDLIST ENA = 1 then
issue SOURCE ADDRESS = DATA SOURCE ADDRESS
issue TRANSFER LENGTH = DATA TRANSFER LENGTH
else
issue SOURCE ADDRESS = D SOURCE ADDRESS
issue TRANSFER LENGTH = D TRANSFER LENGTH
end

issue DESTINATION ADDRESS = D DESTINATION ADDRESS
issue TRANSFER TYPE = D TRANSFER TYPE
issue TRANSFER CONFIGURATION = D TRANSFER CONFIGURATION

Subunit 3.1 'Define Transfer Length'

This subunit is used to choose a suitable transfer length for linked list transfer based on the Linked List mode. When Dynamic Linked List occurs, the corresponding dynamic transfer length 'Data Transfer Length' is chosen. The input and output signals can be seen in Table 5.7.

Input	Output
D Transfer Length	linkedlist Transfer Length
Data Transfer Length	

Table 5.6: Linked List Unit 3 Input and Output

Subunit 3.2 'Define Source Address'

This subunit is responsible to decide the Linked List source address. The process is similar to the Subunit 3.1. The input and output signals can be seen in Table 5.8.

Input	Output
D Source Address	linkedlist Source Address
Data Source Address	

Table 5.7: Linked List Unit 3 Input and Output

Subunit 3.3 'Create Descriptor'

This subunit is used to create the Linked List descriptor and triggers the Core Unit to perform Linked List data transfer process. A 'enable transfer' signal will send to Core Unit when the Linked List descriptor is ready.

Linked List Process 4: Point to Next Descriptor

The 'Point to Next Descriptor' process is designed to link the next descriptor in the Linked List. Inside each descriptor, there is a pointer which points to the next source address of the descriptor. Then the descriptor can be read by the Linked List Unit. The number of the descriptors in the Linked List is fixed before the Linked List transfer occurs. Each Linked List has its corresponding length, and the information about the amount of descriptors is set in the descriptors. The Linked List transfer dose not stop until the last descriptor is executed. The number of executed descriptors is counted in the 'Linked List Status' store. The status is updated right after each descriptor is executed. By comparing the descriptor length and the descriptor count, the Unit 4 decides whether the Linked List transfer continues or not. The result is recorded as 'Linked List Status'. The input and output signals and the corresponding descriptions can be seen in Table 5.9.

Signal	Input/Output	Description
LinkedList Source Address	Input	Start address of Linked List and first descriptor
D Next Descriptor Pointer	Input	Start address of next descriptor
Descriptor Count	Input/Output	The number of executed descriptors
D Descriptor Length	Input	The number of descriptors in the Linked List
LinkedList Status	Input/Output	Current Linked List transfer is on or off.
Descriptor Address	Output	Source address of the next descriptor
Enable LinkedList	Input	Control signal to trigger the Linked List transfer
Finish LinkedList	Output	Control signal to stop the Linked List transfer
Next Descriptor ena	Output	Control signal to read next descriptor

Table 5.8: Linked List Unit 4 Input and Output

Based on the functions of this unit, further functional division is made. The unit is divided into 3 subunits. They are Subunit 4.1 'Define Next Address', Subunit 4.2 'Manage Descriptor Count', Subunit 4.3 'Define LinkedList Status'. The DFD and CFD is developed and can be seen in Figure 5.15 and Figure 5.16.

The operation can be described in the Process Specification.

PSPEC 4

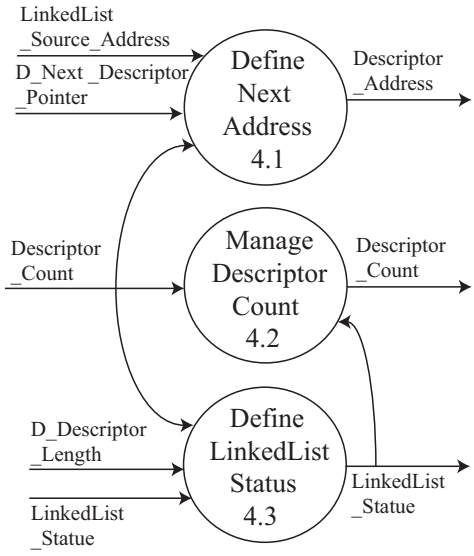

Figure 5.15: Linked List Unit 4 DFD

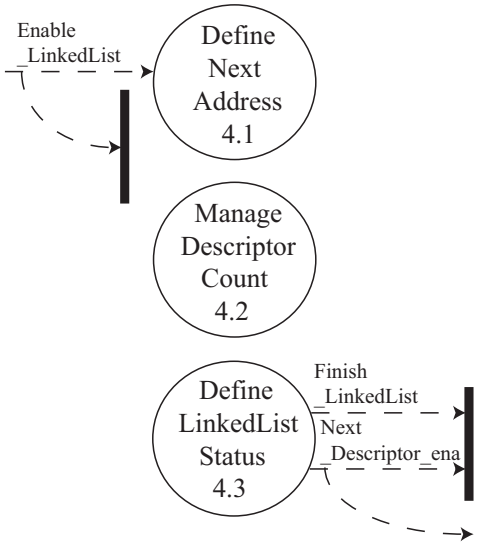

Figure 5.16: Linked List Unit 4 CFD

if LINKEDLIST STATUS = FIRST DESCRIPTOR or FINISHED then
issue DESCRIPTOR COUNT = 1
else
issue DESCRIPTOR COUNT = DESCRIPTOR COUNT + 1
end

if DESCRIPTOR COUNT = 1 then
issue DESCRIPTOR ADDRESS = LINKEDLIST SOURCE ADDRESS
elseif DESCRIPTOR COUNT is larger than 1 and less than D DESCRIPTOR

LENGTH then
issue DESCRIPTOR ADDRESS = NEXT DESCRIPTOR POINTER
else
issue DESCRIPTOR ADDRESS = 0
end

Subunit 4.1 'Define Next Address'

This subunit is designed to define the source address of the next descriptor. When the Linked List transfer just begins, the address is the source address of the first descriptor. Later, each descriptor contains a pointer which points to the next descriptor. According to the requirement, the pointer in the last descriptors must be 0.

Subunit 4.2 'Manage Descriptor Count'

The function of this subunit is to count the number of executed descriptors. The initial value of the descriptor count is 1. Each time a descriptor is read and executed, the descriptor count adds 1. When the count reaches to the descriptor length, the Linked List transfer should be finished.

Subunit 4.3 'Define LinkedList Status'

This subunit is responsible to record the status of the Linked List transfer. The status includes 'firstdescriptor', 'enabled' and 'finished'. The descriptions can be seen in Table 5.10.

Status	Description
firstdescriptor	Begin of the Linked List transfer
enabled	Finish the first descriptor and continue with the remaining ones
finished	All the descriptors have been executed

Table 5.9: Linked List Unit 4 Input and Output

5.3 Conclusions

Once the Core Unit needs to perform Linked List transfer, the descriptor is sent to the Linked List Unit via Core Unit. At the same time, the Core Unit triggers the Linked List Unit to program the Linked List descriptor. The Linked List Unit must support both the Static Linked List mode and Dynamic Linked List mode. In the Logical Architecture, the Linked List Unit is specified into 4 subunits. If the Dynamic Linked List mode is triggered, Subunit 2 takes responsibility to fix the start address and length of the data field of a packet. Subunit 1 is used to read descriptor into the descriptor store. Subunit 3 transforms the descriptor into the Linked List descriptor. After the transform, Subunit 4 points to the address of the next descriptor. Except Subunit1, other functional subunits are further divided to the lowest level by Data Flow Diagram and Control Flow Diagram. And the corresponding input and output signals are proposed. So the Linked List Unit specification is finished.

6 Simulation and Synthesis

The previous chapters present and implement the specification of the Core Unit and the Linked List Unit. In this chapter, the simulations of the Core Unit implementation and corresponding results are discussed first. Synthesis is also done for the Core Unit and is presented after the simulation part.

6.1 Simulation and Results

6.1.1 Simulation Environment Setting

The implementation of Core Unit is done by VHDL programming. The simulation is done through Cadence Simvision. A testbench is set up to take off the real situation. The simulation focuses on the Core Unit implementation code. Since Core Unit is used to transfer data through AHB bus system, a emulator which is able to imitate the function of AHB master should to a certain extent should be developed. Thus, the data can be transferred between Core Unit and the AHB master emulator. By watching the waveforms, the correction of the implementation, as well as the RTL Code can be checked.

The Context Diagram of the simulation environment is designed in Figure 6.1. The descriptor is set to support the burst transfer. After reading address is forward onto the AHB bus, the AHB Master Emulator sends the predetermined read data to Core Unit. Then Core Unit will write the data back to destination address through the Emulator. Control signals, such as clock and channel hready, are also sent to Core Unit to control the data transfer process.

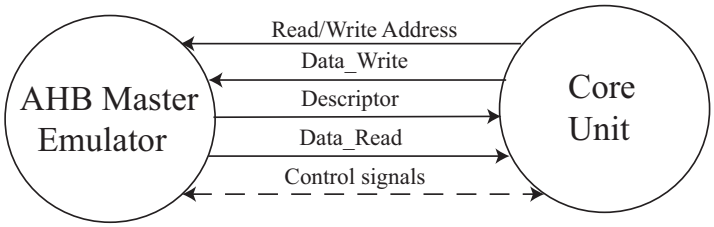

Figure 6.1: Simulation Environment Context Diagram

The simulation uses the test principle, which is set in Chapter 3. The clock frequency is set to be 100 MHz. Since the functions only rely on the implementation, the frequency value has nothing to do with the functions of Core Unit at all. Therefore, other frequency values can be chosen as well. For the practical environment, the signals will not arrive at its destination exactly on the rising edge of clock cycle. Thus, in order to imitate the real system as much as possible, some signals will arrive a litter bit later

than the rising edge of the clock cycle. For instance, the data is read 1ns later than the rising edge. Therefore, when coming to the rising edge, the signals 'data read' and 'data read loaded' is not active until 1ns later.

6.1.2 Simulation Results

From the use cases in Table 6.1 above, there are totally 17 use cases. When DMA Controller is in the zero data element case, the wait states are meaningless. And for the rest of use cases, wait starts can be inserted. And it's not possible to list all the simulation results here. So three typical use cases are chosen to be described in this section. More simulation results can be found in the Appendix C. For each use case, a ideal simulation waveform without time delay is drawn first. This is to show the expected functions of the implementation and also the expected waveform from the EDA tool. Then an analysis is given. The example use case is chosen to be Burst Read State to Burst Write State. The ideal simulation result can be seen in Figure 6.2.

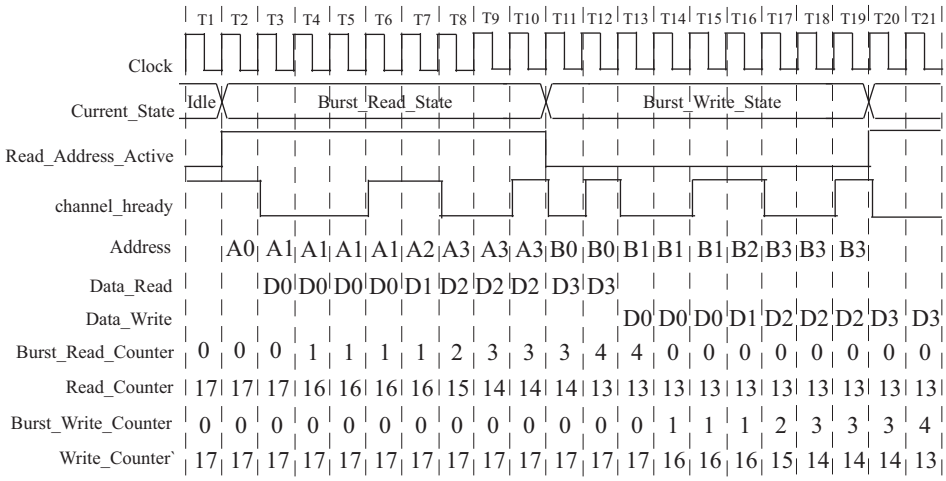

Figure 6.2: Simulation Burst Read to Burst Write

According to AHB protocol, the transfer starts from the address phase clock cycle of the first data element, finished at the data phase clock cycle of the last data element. Thus, the amount of clock cycle consumed in the transfer can be calculated. The number of data elements in a burst is set to 4. So for the burst transfer without wait state insertion, it takes 9 clock cycles to transfer one burst of data. When the wait states are inserted into the state, the situation becomes complex.

The 'channel hready' controls the wait state. When it becomes low, the wait states starts to be inserted into the transfer process. Wait states do not stop until the signal returns to high again. According to AHB protocol, the 'channel hready' signal comes a little bit later than the rising edge of each clock cycle. As we can see, during Burst Read State, the 'channel hready' signal is high at the rising edge of T2 and T3 periods. So the read addresses A0, A1 are forwarded to the address line continuously. And during T2 period, the data D0 is read by Core Unit. As a consequence, the 'burst read counter' and 'read counter' record the data transfer on the next cycle of the data phase.

The start value of 'read counter' represents the number of required data elements. And each time when a new data is read, the value is decrease 1. On the contrary, the 'burst read address' increases 1. Then at the rising edge of T4, the wait states starts to be inserted. As a result, both the address line and data line is paused. The read address and data stop updating during this period, as well as the counters. At the rising edge of T7 period, the 'channel hready' signal is sampled to be high, and the address and data processes starts to be updated again, as well as the counters. After all the data elements in a burst is read, the 'burst read counter' returns to 0. And the 'read counter' keeps the value till the next data is read. The analysis can also apply to the Burst Write State.

The wait states occurs when one device is slower than the other device. So the data transfer process can obtain enough time to perform data transfer. And since the data is held during the wait states, the data can avoid loosing. But the wait state also has its drawbacks. In this use case, for instance, the clock cycles consumed by the Burst Read State increase from 4 to 9. The speed of the system is decreased. Thus the system becomes less efficient.

The simulation waveform with the same wait states insertion from Simvision window can be seen in Figure 6.3.

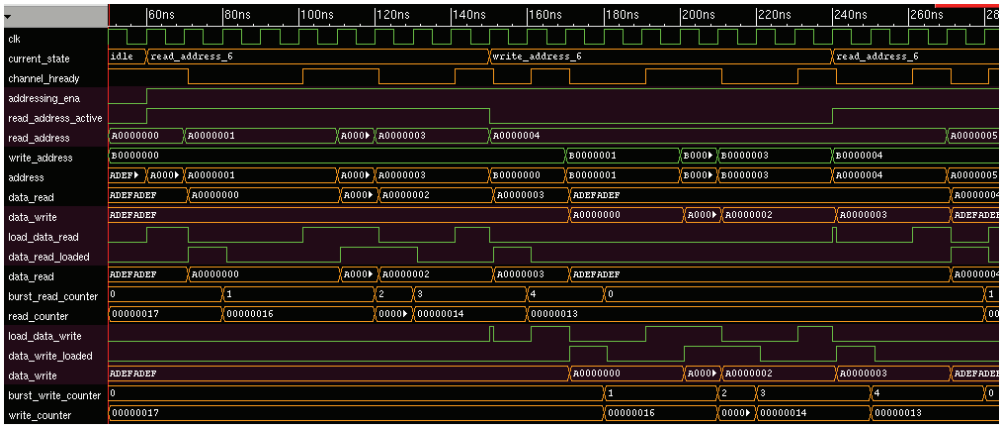

Figure 6.3: Real Simulation Burst Read to Burst Write

6.2 Synthesis Result

A logic synthesis of Core Unit is done through Cadence Ambit. The CMOS technology of the synthesis is 180nm. As a result, the total number of equivalent gates is around 3K. And the total active size is 37k (um2). According to the requirement, the DMA Controller need to have 10 DMA channels. Thus, the number of equivalent gates for 10 DMA channels is 30k. Since the Core Unit is the most complex part of the DMA Controller, so the equivalent gates of the YAD-C DMA Controller is around 50k.

6.3 Conclusions

During the simulation, both the sequential-single and burst transfer is check. And for each transfer type, all the possible cases of wait states insertion are set in the testbench. As a result, the data transfer function of the Core Unit design fulfills the corresponding Core Unit requirements. The synthesis is also done, and the equivalent gates for one Core Unit Cell are 3k.

7 Conclusions and Future Work

7.1 Conclusions

The DMA Controller takes over the data communication from microprocessor and is responsible for the data transfer task in the In-car Entertainment system. Therefore, the microprocessor becomes more efficient in data computation. The DMA Controller in this project must support both normal data transfer and Linked List transfer. The normal data transfer, sequential-single transfer and burst transfer types should be realized. Meanwhile, the DMA Controller should also be flexible for different communication systems, which means the number of DMA channels can be easily increased or decreased according to a particular system. This is realized by using a structure with multiple Core Unit cells. The Core Unit cell is the logical implementation of the DMA channel. Each Core Unit cell is able to perform all the required data transfer types. When cell number changes, the surrounding units, such as channel multiplexer and peripheral multiplexer, do not need to be redesigned. Hatley and Pirbhai methodology is adopted to specify the DMA Controller and the subunits. Following the design flow of this methodology, new functional requirements are proposed. These requirements build foundation maps for the future work. My main efforts are paid on the design of the Core Unit and Linked List unit.

For the Core Unit, the existing results failed to realize the burst transfer with wait states insertion. By analyzing the existing results, I find that the specification of the Core Unit was wrong. The subunit 1, which was responsible for the address and counter management, was not well defined. Meanwhile, the state machine of subunit 2 is only based on the one-cycle wait states insertion. In fact, wait states can last for more than one cycle according to the conception of the wait states. I redo the specification of the Core Unit, as well as the state machine of the subunit 2. The simulation indicates the Core Unit is able to perform all the required transfer types, especially for the burst transfer with multiple wait states insertion. The Core Unit cell is also synthesized and the number of the equivalent gates for one Core Unit cell is 3k.

For the Linked List Unit, I finish the specification of the Linked List Unit for both Static Dynamic Linked List mode. All the functional subunits are described in both Data Flow domain and Control Flow domain. The corresponding Data Flow Diagram and Control Flow Diagram are drawn. The input and output signals of each functional units are indicated. Based on these signals, the process specifications are designed using structure English. The results of all the specifications in different level provide a clear map for the future implementation.

7.2 Future Work

There are several recommendations for the following internship student who decides to continue with this project. The recommendations are divided into several sections of the project.

Linked List Implementation

The Linked List Unit is fully specified. The process specification indicates the principles of the state machine. Data Packing function is very important for the Dynamic Linked List transfer because of the data formats from different peripheral.

Flow Control Unit

Flow Control Unit is responsible to control the handshake signals. This is necessary for the DMA transfer between DMA Controller and peripherals.

Appendix A

Introduction of Hatley and Pirbhai Method

Hatley and Pirbhai methodology is an object-oriented hierarchical design methodology used in the Real-Time Embedded System design. Hatley and Pirbhai methodology starts from setting the function requirements of a system or a IP block. The requirements are a set of independent statements that specify the problems that the system is going to solve. These problems can be seen as a set of independent functions required by the system. Therefore, at the very beginning of the design, function requirements should be specified.

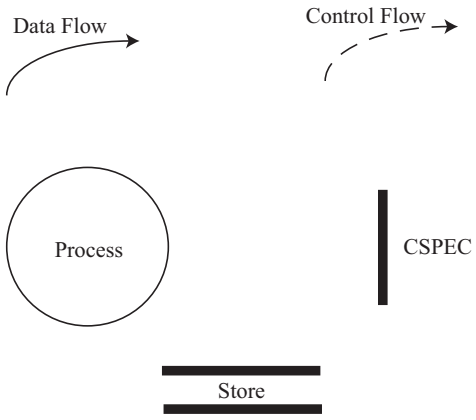

Figure A.1: Hatley and Pirbhai Elements and Symbols

The highlight of Hatley and Pirbhai methodology is functional partitioning in terms of Data Flow and Data Process. This is done by using Data Flow Diagrams (DFD). The Data Flow represents the inputs and outputs data. And the Data Process transforms one input into its corresponding output. In the thesis project, the Data Process is the functional unit. The Data Flow Diagram imitates the operation of the system in terms of a network of processes connected by a group of Data Flow. The Data Process is built from the top level, which represents the top level function of the system. Then, the process is partitioned into a set of subprocesses. Each process holds a simpler subfunction. Thus, through functional partitioning, a complex system can be divided into a set of simple function units. The design can be started from a simple function realization. Each DFD has its own Control Flow Diagram (CFD), which is in terms of Control Flow and Data Process. The Control Flow influence the behavior of the Data Process by enabling or disenabling some functions. The Process Specification (PSPEC) used in the Chapter 5, shows the transformation of Data Flows in a clear, unambiguous and non-redundant manner [6]. The state machines can be seen as Control Specification (CSPEC), which shows the transformation of control flows to model the

activation of processes on DFD/CFD [6]. Both the Data Flow Diagram and the Control Flow Diagram consists of five elements. Some of the elements and their corresponding symbols, which are used in the thesis project, can be seen in Figure A1. The Store shown in Figure A1 in hardware domain can be seen as memory, which stores the data flows. For more details, book “Strategies for Real-Time System Specification” is recommended, which is written by Hatley, Derek J. and Pirbhai, Imtiaz A..

Appendix B

Simulations of Data Streaming Methods

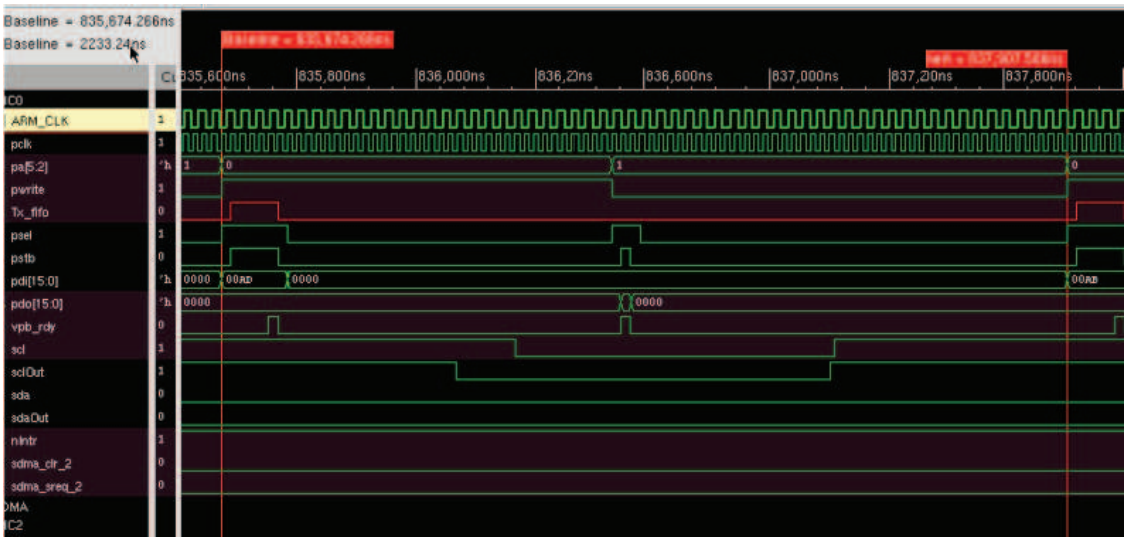

Figure B.1: Polling-Based Transfer

Figure B.2: Interrupt Based Transfer

Figure B.3: DMA Based Transfer-Initial DMA Operation

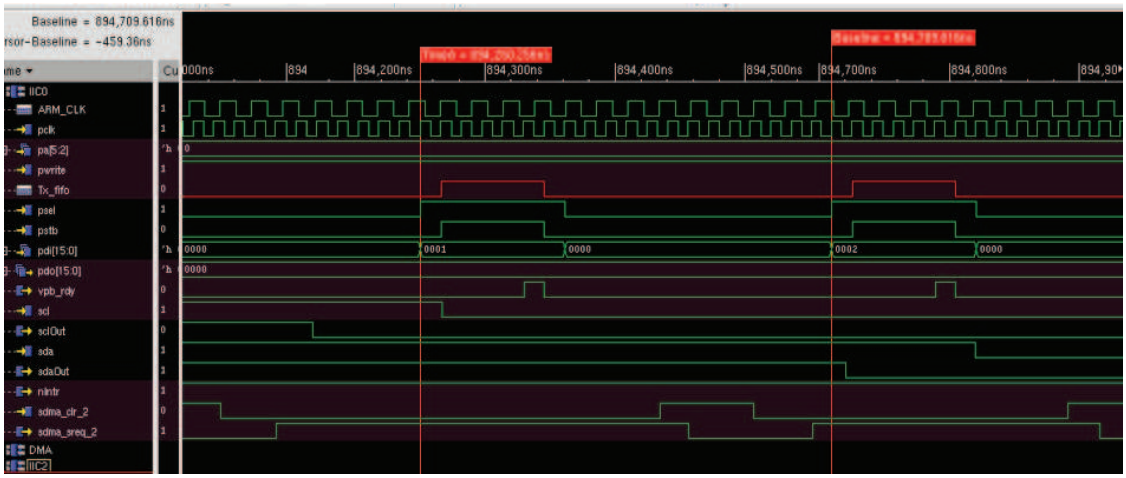

Figure B.4: DMA Based Transfer-DMA Transfer

Appendix

Examples of Simulation Waveforms

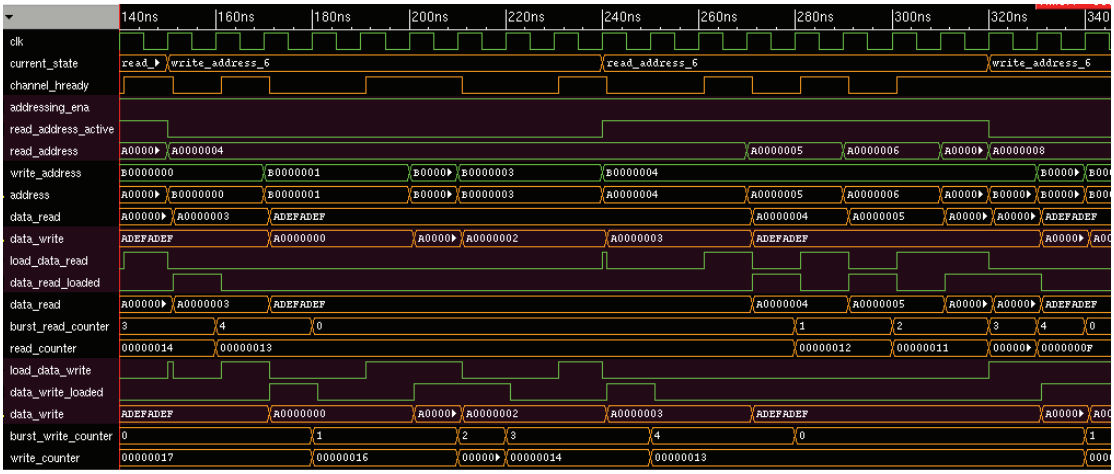

Figure C.1: Burst Write State Transforms to Burst Read State

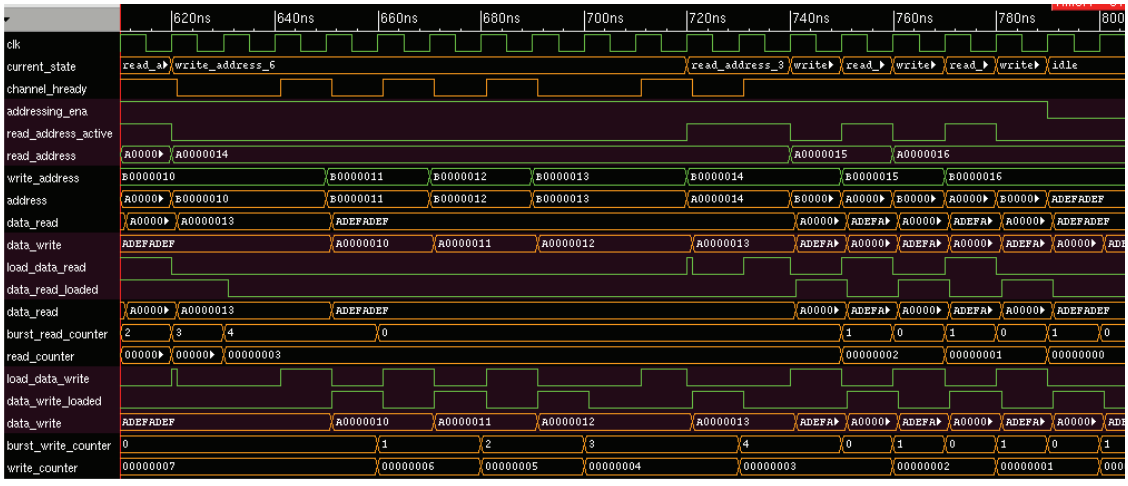

Figure C.2: Burst Write State Transforms to Sequential-single State

Bibliography

[1] Maarten Scheuter, "Yad-c dma-controller functional requirement specification," Tech. Rep., NXP Semiconductors, 2006.

[2] Nicolas Rihet, "Design and implementation of a new dma controller," Tech. Rep., NXP Semiconductors, 2008.

[3] *Fundamentals of Digital Logic and Microcomputer Design*, John Wiley and Sons, Inc., 2005.

[4] *An Introduction to Digital Computer Design*, Prentice-hall of India Pvt Ltd., 2008.

[5] "Preliminary ic requirement specification of one chip carradio circuit," Tech. Rep., NXP Semiconductors, 2006.

[6] "Structured analysis and structured design for real-time systems," Tech. Rep., Philips, 1998.