ASIC for Navigation Application with Programmable Peripheral Logic

PROJECT REPORT

Submitted by

MANJUSHA M

TVE16EC032

RAMDAS PRASAD H

TVE16EC040

SRUTHI SANTHOSH

TVE16EC051

to

the A P J Abdul Kalam Technological University

in partial fulfillment of the requirements for the award of the Degree

of

Bachelor of Technology

In

Electronics and Communication Engineering

Department of Electronics and Communication Engineering

College of Engineering

Trivandrum

July, 2020

DECLARATION

We undersigned hereby declare that the project report (“ASIC for Navigation Application with Programmable Peripheral Logic”) , submitted for partial fulfillment of the requirements for the award of degree of Bachelor of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by us under supervision of Prof. Lalu V. This submission represents our ideas in our own words and where ideas or words of others have been included, we have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Manjusha M

Place

Date

Ramdas Prasad H

Sruthi Santhosh

DEPT. OF ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING

TRIVANDRUM

2019 - 2020

CERTIFICATE

This is to certify that the report entitled **ASIC for Navigation Application with Programmable Peripheral Logic** submitted by **Manjusha M, Ramdas Prasad H and Sruthi Santhosh**, to the APJ Abdul Kalam Technological University in partial fulfillment of the B.tech. degree in Electrical and Electronics Engineering is a bonafide record of the project work carried out by them under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Project Coordinator	**Project Coordinator**	**Head of the Department**
Prof. Gijy P. G.	Prof. Lalu V.	Dr. Santhosh Kumar S.
Assistant Professor	Assistant Professor	Professor
Dept. of ECE.	Dept. of ECE.	Dept. of ECE.
College of Engineering	College of Engineering	College of Engineering
Trivandrum	Trivandrum	Trivandrum

ACKNOWLEDGEMENT

We take this opportunity to express our deep sense of gratitude and sincere thanks to all who helped us to complete the work successfully. Our first and foremost thanks go to God Almighty who showered in immense blessings on our effort.

We wish to express our sincere thanks to **Dr. Jiji C. V., Principal, College of Engineering, Trivandrum** for providing us with all the necessary facilities and support.

We would like to express our sincere gratitude to **Dr. Santhosh Kumar S (Head of the department)**, for his support and co-operation. We wish to express our sincere gratitude towards all the teaching and non teaching staff members of our Department.

Finally We thank our parents, all our friends, near and dear ones who directly and indirectly contribute to the successful completion of this work.

Manjusha M
Ramdas Prasad H
Sruthi Santhosh

ABSTRACT

Systems designed for navigation require processing of real-time and conventional computing. This implies that the control hardware must be easily customizable, easy to debug, reliable and capable of handling real-time data, which may change depending on the application. These constraintsare typically met through two seperate solutions. A microcoontroller to handle control tasks and an FPGA to handle rela-time data. This is the optimal solution to the problem and is the the most widely accepted solution that currently exists. This project aims to integrate these two solutions onto a single Application Specific Integrated Circuit(ASIC) that meets all requirements. The Compute core, and programmable peripherals will be built up from scratch, along with essential core peripherals like UART and MIL-1553-STD. All other necessary protocols will be sourced from FPGA/ASIC proven open source, wishbone compatible peripherals. The aim of this project is create a roadmap for this special class of ASICs that have great value for the military as well as scientific application.

CONTENTS

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

1.1 GENERAL BACKGROUND

Many real time applications where control Systems are utilized benefit from having hardware that can handle real time data. In situations where large-scale production of Application Specific Integrated Circuits (ASICs) for each and every problem situation is not feasible most conventional design requirements are met using Field Programmable Gate Arrays (FPGAs). This approach yields the benefit of the same template hardware boards to be reconfigured as per the specific application, even on the fly and give considerable flexibility to the platform while also contributing heavily to the considerable cost of the overall design. Hence for applications like Navigation systems for satellites the benefits of both approaches can be combined to create an ASIC which includes a processing core, multiple configurable logic blocks, support for standard IO interface standards and protocols and ADC/DACs, all on the same silicon. This allows considerable benefits to help tailor custom hardware, specific to the needs where the system is to deployed while allowing for the reliability and cost-effectiveness of Large scale production of ASICs in house.

RISC-V is an Open-Source, frozen, Instruction-set Architecture (ISA), initially developed at The University of California, Berkeley. The ISA is a becoming a vastly popular alternative to CISC architectures and proprietary popular RISC architectures like ARM. The applications the system will encouter warrants the use of RISCV32-IM, which is the integer operation base 32 bit variant of the RISC-V ISA, with the multiply/divide extension.The processing core should preferably be pipelined and have necessary hazard detection and mitigation schemes.

The inclusion of FPGA Blocks on the die presents unique software and hardware challenges that require adapting known standard solutions to function in the unique environment the ASIC design creates.These include adapting bus standards for ease of communication with and programming of the FPGA. Establishing optimal routing for the given FPGA architecture and automating the process of synthesizing a design on the FPGA blocks using the processing core, handling and generation of multiple clock domains internally and ability for the SoC to drive output pins to different output voltage standards according to the application. This also brings about the need for an integrated onboard or external power management solution and multiple IO voltage domains. The project aims to create a unique ASIC that currently does not exist in the market and can find application in multiple domains.

1.2 OBJECTIVE

1.3 SCOPE

The project will have four different goals to achieve based on the ideas presented in the preceeding section. Firstly, Design a usable RISC-V compute core that is flexible and light-weight. Secondly implement the Programmable FPGA cores and implement a programming scheme. and then implement the necessary peripherals. Thirdly, synthesise and test the SoC elements on an FPGA and lastly, tapeout and fabrication. All elements of the SoC will be written in chisel and exported to verilog. Verilator can be used to test the SoC at the verilog level and exported to the tapeout tools available at the IISU before final fabrication.

1.4 SCHEME OF PROJECT WORK

The RISC-V Instruction-set Architecture is highly pipelining friendly as well as being comparatively easier to implement in hardware than nore conventional RISC ISAs like MIPS. This enables the design of a fast, minimalistic, pipelined compute core that can easily work as a backbone for the additional components necessary to meet the application requirements. It follows that a flexible yet simple FPGA architecture needs to be identified keeping in mind both the ease of ease of implementation and ease of programming it by the core. The Navigation ASIC will encounter both Real-Time Data and Complex Decision making tasks. Hence the compute core and the FPGA elements have distict roles to play in the ASIC. The FPGA block can be programmed to have Logic that handles the real time data while the compute core can handle less time-constrained tasks. To take it one step further the FPGA can be programmed on the fly by the compute core, thus ensuring better flexibility for the ASIC as well as making it much more of a powerful solution, eventhouh the individual components are not by any stretch, new architectural elements.

1.5 Constraints

The proposed SoC should contain the following elements:

- RISC-V ISA processor with a five stage pipeline, hazard detection and avoidance
- UART, SPI(boot and programming through this interface), I2C, MIL-STD 1553 encoder decoder.
- On-board ADC and DAC interfaces.
- Core Configurable FPGA Logic Blocks with multiple clock domain routing.
- Pin Drive Circuitry and Voltage level shifting.

1.6 Proposed process

According to the resources available the preffered HDL for design has been determined to be a combination of verilog and chisel. The design will be tested on an FPGA and once validated the design can be prepared for layout and fabrication. The rest of the project will be implemented using the Cadence Software Toolchain. The design process can be summarized in the following steps:

1. Creating a behavioural model of the ASIC in verilog using Chisel.
2. Testing the design using a Verilog simulator like verilator.
3. Exporting the design to FPGA and testing the ASIC subcomponents.
4. Exporting the design to Cadence toolchain for onboarding and layout tasks.
5. Preparing ASIC for fabrication using SCL 180nm as the target and testing.
6. Mask generation and fabrication.

CHAPTER 2

System Architecture

2.1 Hardware Architecture

The Architecture of the ASIC should provide the optimal balance of flexibility and reliability for the scenarios it is being designed for. This chapter is dedicated to the detailed architecture of the proposed ASIC

2.1.1 Core Design

ALU with add, multiply, subtract, division, and, or, not, xor, operation support. Five stage pipeline: Instruction Fetch(IF), Instruction Decode(ID), Execution/address calculation (EX), Memory fetch or write(MEM), Write Back(WB). Hazard Detection and avoidance: Forwarding, stalling and branch prediction(assume always dropped unless destination address precedes PC, 32 registers in register file (std),

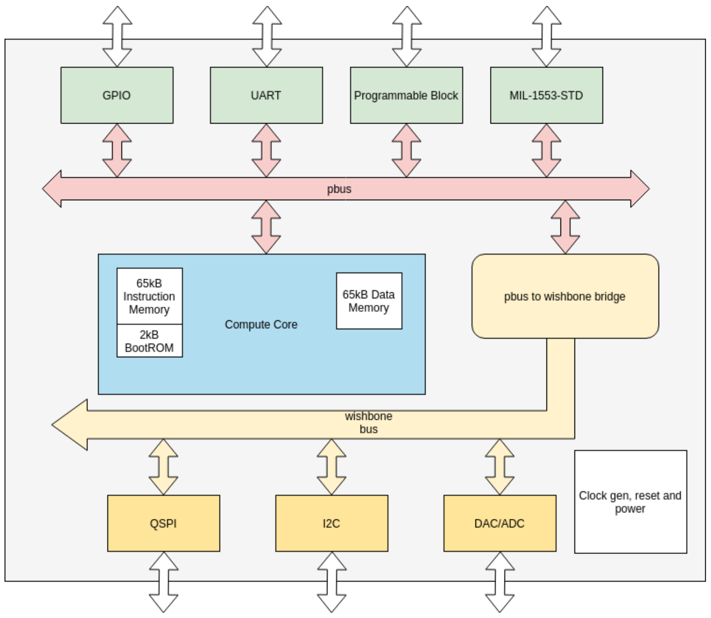


Figure 2.1: Navigation AISC Base Architecture Block Diagram

CHAPTER 3

Detailed Design

3.1 Compute Core

The compute core is a five stage pipled RV-32IM processor as described in previous sections. This chapter illuastrates the functional blocks thatr comprise the core and their functionality.

3.1.1 Instruction Fetch stage

The Instruction Fetch(IF) stage is tasked with the reading and writing to Instruction memory, Program Counter updtaion and handling of pipeline stalls. Each of these tasks is handled in the following ways:

1. The **PC** register is updated always to increment by a value of 4 (corresponding to the address spcae of 1 32-bit instruction) each clock cycle. This value is used as the fetch address for the instruction memory.The instruction is formed out of the **dataOut** lines in the instruction memory. The architecture expects a Memory configuration with read latency of one clock cycle.

2. Due to a read latency of one clock cycle the **PC** value is stored in anouther register **PCout**, from where the **PC** value is fed to the decode stage. Branch and stall operations account for this delay in fetch as well.

3. Pipeline stalls require atleast two clock cycles to load the output of the stage, hence all stalls will be a minimum of two clock cycles. during these the output instruction is an 'nop' instruction or 0x00000013.

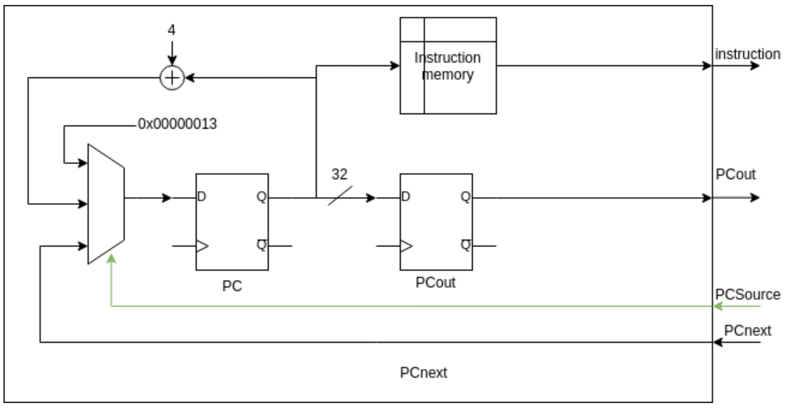


Figure 3.1: Instruction Stage Block Schematic

A block schematic is illustrated in Fig.3.1.The IF stage can be programmed for debugging using the first of two programming ports. The programming is done by halting the pipeline using the **io_halt** line. The second read write port is mapped to a higher read address than the data memory. Refer memory map and archuitecture for more details.

3.1.2 Execution Stage

The Execution stage is, as the name implies, tasked with the execution of the instruction. The functional block diagram of the Stage is given in Fig.3.2.

ALU

As per the requirements of the Compute core as described in the previous chapters, it is clear that an effective ALU must be simple, and support maximum hardware opeartions with minimal hardware utilization. Hence each possible opeartion the ALU can perform in hardware directly through a behavioural model is given a unique identifier signified by

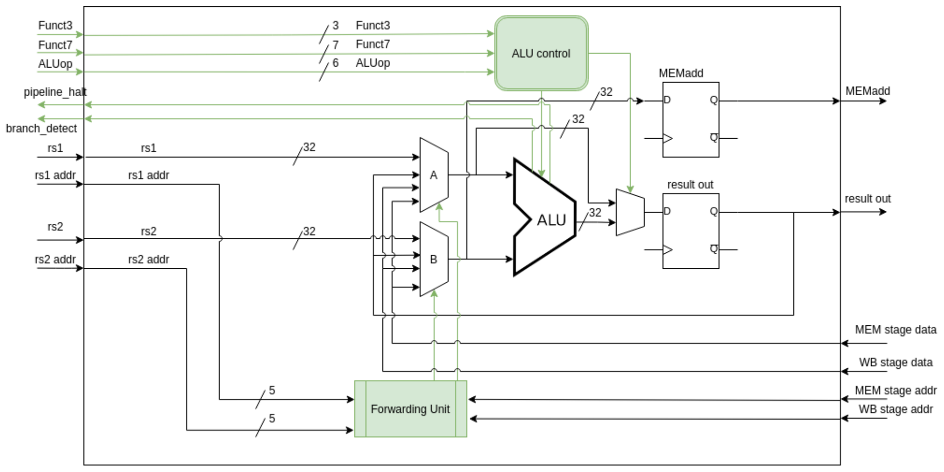


Figure 3.2: Execution Stage Block Schematic

the bits of the **ALUctl** signal except the Most significant bit and secod-most significant bit. These bits explicitly describe the suboperation on the hardware to be performed. For example MUL, MULH, MULHSU and MULHU share the same **ALUctl[4:0]** value of 0101. The operations and their corresponding **ALUctl** values are summarized in Table 3.1

ALUControl

The **ALUctl** values specified in Table 3.1 can be used in conjunction with the RISCV instruction set user level encoding to generate a truth table for a peice of logic called the ALUControl. This Logic is tasked with decode the operation the ALU has to perform based on the Operation type specified by instruction using a signal generated by the Control Logic and additional Funct7 and Funct3 bits. this decoding is summarized in Tables 3.2 and 3.3

Using the information in Table **??** the values for all insputs for which each output bit of the **ALUctl** values can be extracted and the following output equations can be derived, assuming **ALUop** is represented by **a**, **Funct7** by **f** and **Funct3** by **b** as shown by Equations 3.1 to 3.6. They are heuristically reduced to a minimum size and implemented using

Operation	ALUctl
and	000000
or	000001
xor	000010
lt	000011
ltu	010011
add	000100
sub	010100
mul	000101
mulh	100101
mulhu	110101
mulhsu	010101
div	000110
divu	010110
rem	000111
remu	010111
sll	001000
srl	011000
sra	111000

Table 3.1: ALU Operations with their corresponfing ALUctl line values

a purely combinational circuit using gate based logic and it behaviourally.

$$\begin{aligned} ALUctl[0] = & a'_3a_2a'_1a'_0b_2 + a'_3a_2a_1a_0b'_2b_1 + a'_3a_2a_1a_0b_2b_1b'_0 + \\ & a'_3a_2a_1a'_0f'_6f'_5f'_4f'_3f'_2f'_1f'_0b'_2b_1 \\ & + a'_3a_2a_1a'_0f'_6f'_5f'_4f'_3f'_2f'_1f'_0b_2b_1b'_0 \\ & + a'_3a_2a_1a'_0f'_6f'_5f'_4f'_3f'_2f'_1f_0b'_2b'_1 + a'_3a_2a_1a'_0f'_6f'_5f'_4f'_3f'_2f'_1f_0b_1 \end{aligned} \tag{3.1}$$

$$\begin{aligned} ALUctl[1] = & a_3'a_2a_1'a_0'b_2 + a_3'a_2a_1a_0b_2'b_1 + \\ & a_3'a_2a_1a_0b_2b_1'b_0' + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0'b_2'b_1 + \\ & a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0'b_2b_1'b_0' + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0b_2 \end{aligned} \tag{3.2}$$

$$\begin{aligned} ALUctl[2] = & a_3'a_2' + a_3'a_2a_1b_2'b_1' + a_3'a_2a_1'a_0b_2'b_1b_0' + a_3'a_2a_1'a_0b_2b_1' \\ & + a_3a_2'a_1'a_0'b_2'b_1' + a_3a_2'a_1'a_0'b_2'b_1b_0' + a_3'a_2a_1a_0b_2'b_1'b_0' \\ & + a_3'a_2a_1a_0'f_6f_4'f_3'f_2'f_1'f_0'b_2'b_1'b_0' + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0 \end{aligned} \tag{3.3}$$

$$ALUctl[3] = a_3'a_2a_1f_6'f_5'f_4'f_3'f_2'f_1'f_0'b_2'b_1'b_0 + a_3'a_2a_1f_6'f_4'f_3'f_2'f_1'f_0'b_2b_1'b_0 \tag{3.4}$$

$$\begin{aligned} ALUctl[4] = & a_3'a_2a_1'a_0'b_2'b_1' + a_3'a_2a_1'a_0'b_2b_1 + a_3'a_2a_1a_0b_2b_1b_0 + \\ & a_3'a_2a_1a_0f_6'f_4'f_3'f_2'f_1'f_0'b_2b_1'b_0 + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0'b_2'b_1'b_0' + \\ & a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0'b_2'b_1b_0 + a_3'a_2a_1a_0'f_6'f_4'f_3'f_2'f_1'f_0'b_2b_1'b_0' + \\ & a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0b_2'b_1 + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0b_2b_0 \end{aligned} \tag{3.5}$$

$$ALUctl[5] = a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0b_2'b_0 + a_3'a_2a_1f_6'f_5f_4'f_3'f_2'f_1'f_0'b_2b_1'b_0 \tag{3.6}$$

Forwarding Unit

The forwarding unit is tasked with ensuring that back-toback instructions that write and read to the same memmory location and/or the same register is handled smoothly in the pipeline. It functions by routing values from subsequent pipeline stages. Under these considerations three possible forwarding paths come up that the Forwarding Unit has to support. These are:

- When a previously computed output value becomes the current input argument to the ALU
- When the destination register of the Memmory Fetch stage is the argument of the subsequent ALU operation.
- When a a register who is being written to in the Write Back stage is the argument for a simultaneous ALU operation.

Branch Detection

The branch detection Unit is tasked with Confirming that a Branch condition is valid and the branch can be taken. The Branch Predictors prediction is compared and the pipeline is stalled if the they disagree. The branch detction Unit is tasked with making sense of the condition for operations complementary to the Operations supported by the ALU. For example, the sub operation is used to identify wether two registers are equal in case of a BEQ(branch if Equal)instruction by just checking if the output of the ALU is zero, if not, the brach is dropped. The Logical Inverse of this operation is required for the BNE(Branch if Not Equal) instruction. This can be achieved by inverting the implications of the compare. If the ouput of the ALU is zero then the branch is dropped, else if the output is non-zero thenthe branch is taken. This can be extended to complementary Operational pairs like BLT and BGE (Branch if Less than and Branch of Greater than or Equal respectively)

3.2 WISHBONE Bus

The WISHBONE System-on-Chip (SOC) Interconnection is a method for connecting IP cores together to form integrated circuits. Open core SOC design methodology utilizes WISHBONE bus interface to foster design reuse by alleviating system-on-chip integration problems.It is much easier to connect the cores, and therefore much easier to create a custom SOC.

Wishbone improves the portability and reliability of the system.The objective behind WISHBONE is that the interfaces should be independent of logic signaling levels that supports both FPGA and ASIC.

It is a flexible interconnection scheme.Since its a standard interface,it can be written using any hardware description language.It supports these features:

- Independent of the application specific functions of the IP cores

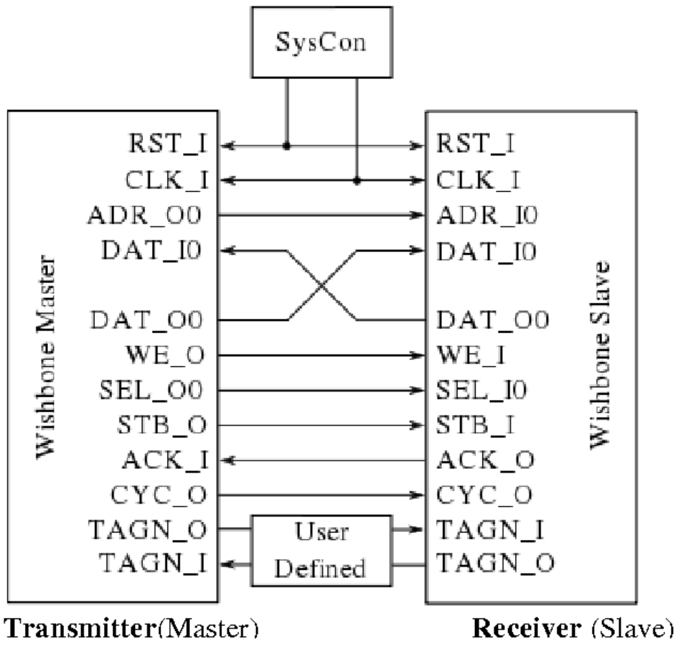


Figure 3.3: Wishbone Bus Interface

- Handles theoretically infinite range of operating frequency
- Supports different types of interconnection architectures
- Absolutely free to use by developers

3.2.1 WISHBONE Basics

WISHBONE utilizes Master and Slave architectures which are connected to each other through an interface called Intercon. Master is an IP core that initiates the data transaction to the SLAVE IP core by providing an address and control signal to Slave. Slave responds to the data transaction with the Master with the specified address range.The Intercon is the medium consists of wires and logics which help in data transfer between Master and Slave. It also requires a SYSCON module which generates WISHBONE reset and clock signal for the system. Figure: 3.4 show the WISHBONE Intercon system which consists of Masters and Slaves and SYSCON modules. The interconnection can be described using hardware description languages like VHDL and Verilog and the design can

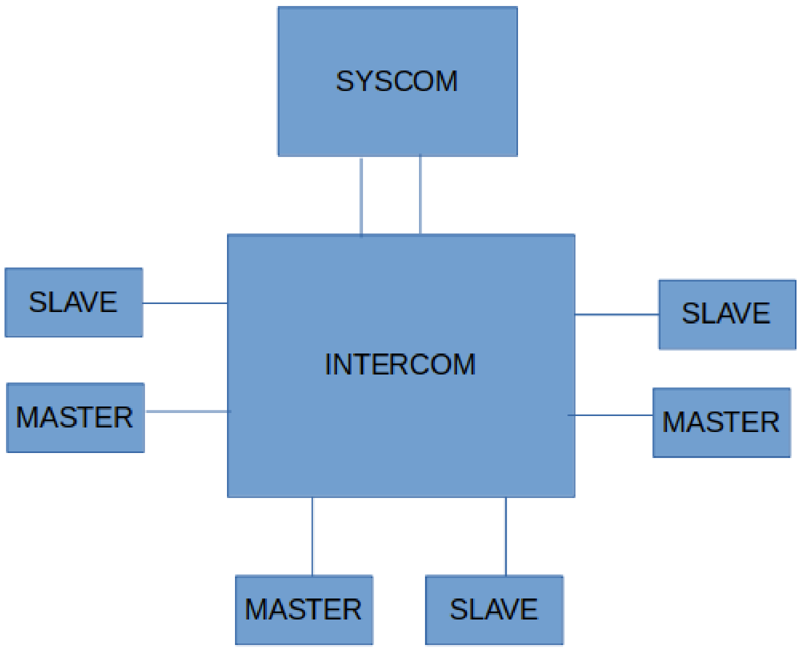


Figure 3.4: Interconnection System

be modified according to requirement.Due to these features,WISHBONE stands different from traditional microcomputer buses.

3.2.2 WISHBONE Features

The WISHBONE interconnection makes System-on-Chip and design reuse easy by creating a standard data exchange protocol. Features include:

- Simple, compact, logical IP core hardware interfaces requires very few logic gates.
- Handshaking protocol allows each IP core to throttle its data transfer speed.
- Supports single clock data transfers.
- Very flexible system designs because of MASTER / SLAVE architecture.
- Synchronous design assures portability, simplicity and ease of use.
- Independent of hardware technology (FPGA, ASIC, etc.).

The WISHBONE specification regulates the ordering of data. This is because data can be presented in two different ways. In the first way, the most significant byte of an operand is placed at the higher address. In the second way, the most significant byte of an operand can be placed at the lower address. These are called Big Endian and Little Endian data operands, respectively. WISHBONE supports both types.

3.3 MIL STD 1553

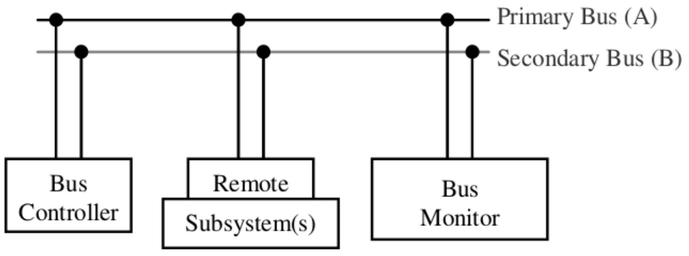


Figure 3.5: MIL STD 1553

The Hardware elements of MIL-STD-1553 Bus consist of the data bus and terminals. The three terminals are:

1. Bus controller (BC)
2. Remote terminal(RT)/Remote Subsystems
3. Bus Monitor Terminal (MT)

Remote Terminal

A remote terminal typically consists of a transceiver, an encoder/decoder,a protocol controller, a buffer or memory, and a subsystem interface.

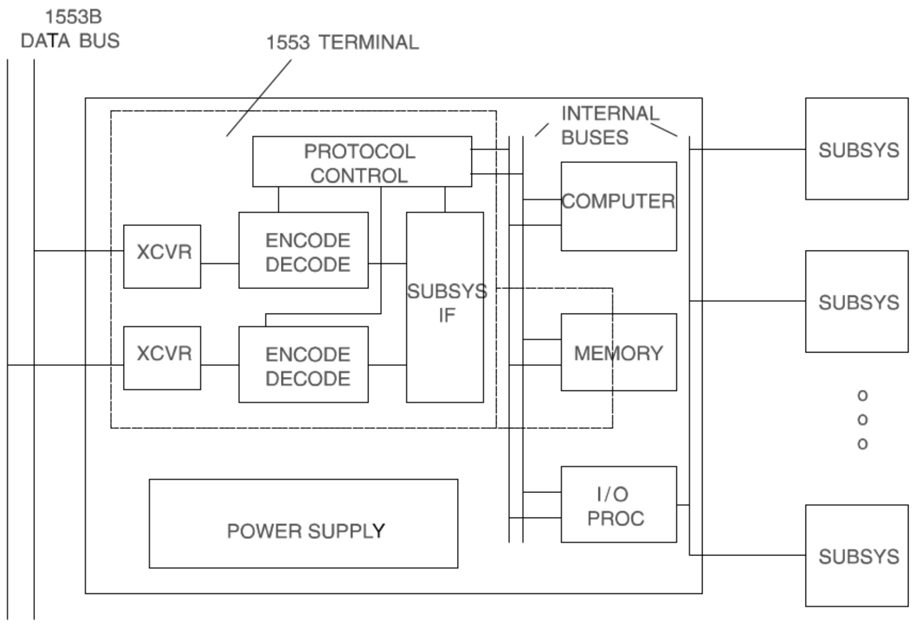


Figure 3.6: Terminal Definition

The Remote Controller is capable of receiving and decoding commands from the bus controller and responding accordingly. It is also be capable of buffering a message worth of data, detecting transmission errors and performing validation tests upon the data, and reporting the status of the message transfer. A remote terminal must follow the protocol defined by the standard. It can respond to commands received from the bus controller. When it receives a valid command, it must respond within a very small, closely defined amount of time.If it is not met, then the remote terminal must invalidate the message and discard the data. It is also capable of reporting status to the bus controller.

Bus Controller

The bus controller is responsible for directing the flow of data on the data bus. While several terminals may be capable of performing as the bus controller, only one bus controller may be active at a time. The bus controller is the only one allowed to issue commands for the transfer of data or the control and management of the bus onto the data bus.The complexity of the electronics associated with the bus controller is dependent of the subsystem interface, the amount of error management and processing to be performed, and the architecture of the bus controller.

Bus Monitor

A bus monitor is a terminal that monitors the exchange of information on the data bus.A monitor may collect all the data from the bus or may collect selected data. Bus monitors fall into two categories:

1. A recorder for testing.
2. A terminal functioning as a back-up bus controller.

In collecting data, a monitor must perform the same message validation functions as the remote terminal and if an error is detected, inform the subsystem of the error. It is common for bus monitors to contain a remote terminal. When the monitor receives a command addressed to its terminal address, it responds as a remote terminal. For all other commands, it functions as a monitor. The remote terminal portion can be used to provide feedback to the bus controller, which monitors status and the amount of memory or time left.

Operation

The Standard defines the data bus to be a single path among the bus controller and remote terminals.MIL-STD-1553B defines the data bus structure for interconnection of up to 31 remote terminal (RT) devices.

A single controller device on the bus initiates the command/response communication with the remote devices. The remote and control devices are interconnected over two, separate buses.Normal operation involves only the primary bus.The secondary bus is available as redundant backup in the event of primary bus damage or failure.The BC is the master device and operating as a bus controller. The BC initiates all the information transfers through the data bus.The standard specifies the information transfers between RTs to follow a command/response format.The BC sends commands to the RTs to tell them what to do. The BC can be a separate subsystem or just a portion of a subsystem.The RT is the device interfaces the data bus to the sub-system and transfers data in and out of the subsystem.

The Standard allows for up to 31 RTs in a system.The Bus Monitor Terminal is optional and works like a passive device which examines all data on the bus. It can record all data or selected data for off-line applications

MIL-STD-1553 communication uses three word types:

1. Command
2. Status
3. Data

All these three words are of 20 bits in length.First three bits are used for the word sync, 16 bits are used for information and the last one bit is for parity. The sync bit differentiates the data words from command and status words.In addition, mode messages are defined for managing the bus system and error recovery.

3.3.1 Command Word

The BC issues Command Word (CW) to RTs to perform specific functions. The CW is shown in the Figure 3.8.The address field is 5 bit and the BC can address maximum 31 terminals.Command words are issued/transmitted only by the BC. BC directs any RT to transmit, receive or perform a specified action.

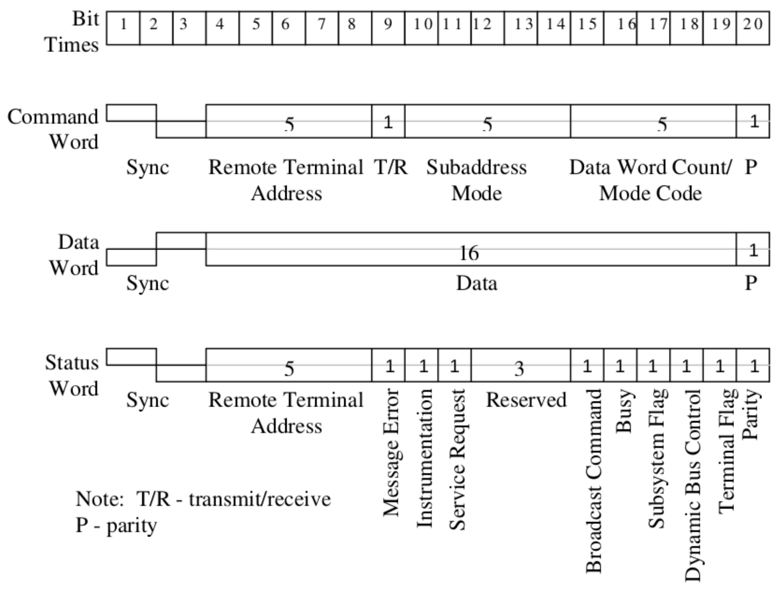


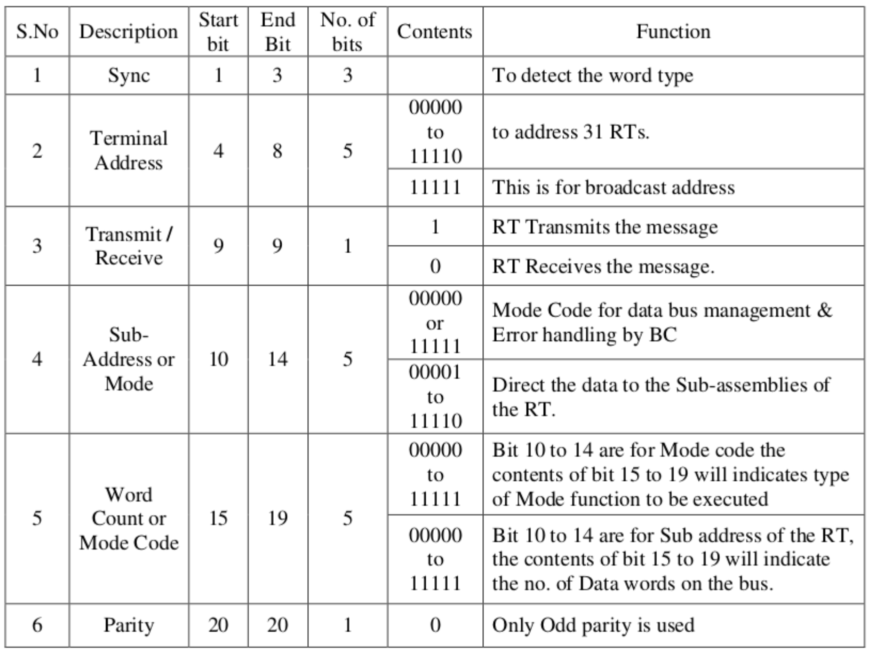
Figure 3.7: Word Formats

3.3.2 Status Word

A Remote terminal(RT)sends Status word in response to BCs command word. The status word is to convey that the transmission is OK or error.The status word is issued/transmitted only by a RT and provides general information on the state of the RT as shown in Figure 3.9

3.3.3 Data Word

The Data Word (DW) contains the actual information that is to be transferred within the message.Data words may be transmitted by either the BC or RT.Messages are from BC to RT transfers, RT to BC transfers, and RT to RT transfers.The first three-bits are “data sync” as shown in Figure 3.10.



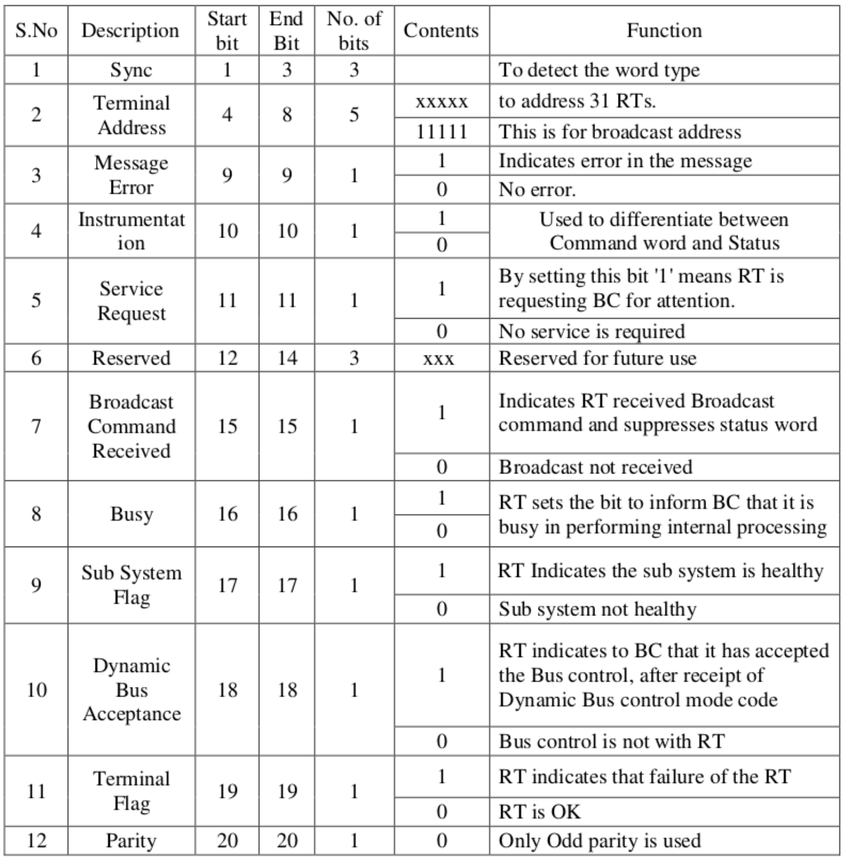
S.No	Description	Start bit	End Bit	No. of bits	Contents	Function
1	Sync	1	3	3		To detect the word type
2	Terminal Address	4	8	5	00000 to 11110	to address 31 RTs.
					11111	This is for broadcast address
3	Transmit / Receive	9	9	1	1	RT Transmits the message
					0	RT Receives the message.
4	Sub-Address or Mode	10	14	5	00000 or 11111	Mode Code for data bus management & Error handling by BC
					00001 to 11110	Direct the data to the Sub-assemblies of the RT.
5	Word Count or Mode Code	15	19	5	00000 to 11111	Bit 10 to 14 are for Mode code the contents of bit 15 to 19 will indicates type of Mode function to be executed
					00000 to 11111	Bit 10 to 14 are for Sub address of the RT, the contents of bit 15 to 19 will indicate the no. of Data words on the bus.
6	Parity	20	20	1	0	Only Odd parity is used

Figure 3.8: Command Word

3.4 QUAD SPI

The quad SPI flash controllers manages the accesses and queries that are sent and received from the SPI device provided with additional two data lines and a enabling mode to operate all the four data lines concurrently and codirectionally. The QSPI mostly interacts using normal SPI protocols while some commands works based on higher level 4 bits.Here the underlying operation of the flash device is masked by the bus controller behind a wishbone interface, so that reads and writes appear simple as when using wishbone interface alone.In QSPI,reads from device is similar to wishbone reads, writes programs the device only if the write protect is properly removed and needs some kind of working with wishbone , while erase commands are controlled using particular registers.The main advantages of QSPI are ,they can provide approximately four times throughput tha normal SPI and multiple devices can be connected to QSPI at the same time.

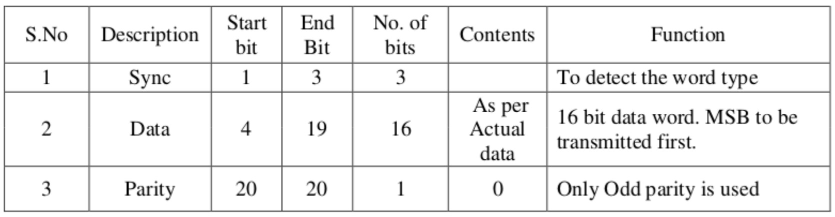
3.4.1 QSPI Architecture



S.No	Description	Start bit	End Bit	No. of bits	Contents	Function
1	Sync	1	3	3		To detect the word type
2	Terminal Address	4	8	5	xxxxx	to address 31 RTs.
					11111	This is for broadcast address
3	Message Error	9	9	1	1	Indicates error in the message
					0	No error.
4	Instrumentation	10	10	1	1	Used to differentiate between Command word and Status
					0	
5	Service Request	11	11	1	1	By setting this bit '1' means RT is requesting BC for attention.
					0	No service is required
6	Reserved	12	14	3	xxx	Reserved for future use
7	Broadcast Command Received	15	15	1	1	Indicates RT received Broadcast command and suppresses status word
					0	Broadcast not received
8	Busy	16	16	1	1	RT sets the bit to inform BC that it is busy in performing internal processing
					0	
9	Sub System Flag	17	17	1	1	RT Indicates the sub system is healthy
					0	Sub system not healthy
10	Dynamic Bus Acceptance	18	18	1	1	RT indicates to BC that it has accepted the Bus control, after receipt of Dynamic Bus control mode code
					0	Bus control is not with RT
11	Terminal Flag	19	19	1	1	RT indicates that failure of the RT
					0	RT is OK
12	Parity	20	20	1	0	Only Odd parity is used

Figure 3.9: Status Word

The main core components are the wishbone quad SPI flash controller, wbqspiflash, and the lower level quad SPI driver,llqspi.Wbqspi flash gives commands to the llqspi while it is the llqspi that actually implements the SPI protocol. Inorder to command, **spiWr** line is activated when the device is idle ,that is when the **spiBusy** is low. The data to be sent to lower level driver is placed in the **spiIn** line. The **spiLen** carries a 2 bit value indicating the length of the transactions (8(00),16(01),24(10) or 32 (11) bits). To use all the 4 I/O lines , **spiSpd** is activated to high and **spiDir** is set to 0 for writing and 1 for reading from all the 4 I/Os. **spiValid** line is used to show that the data received from the lower level driver is valid and **spiOut** will contain data with most recently read bits as lower bits.When ever the device is in idle state, the **spiBusy** will go low and next commands will be read.The llqspi checks **spiWr** before each and every transactions and if it is still



S.No	Description	Start bit	End Bit	No. of bits	Contents	Function
1	Sync	1	3	3		To detect the word type
2	Data	4	19	16	As per Actual data	16 bit data word. MSB to be transmitted first.
3	Parity	20	20	1	0	Only Odd parity is used

Figure 3.10: Data Word

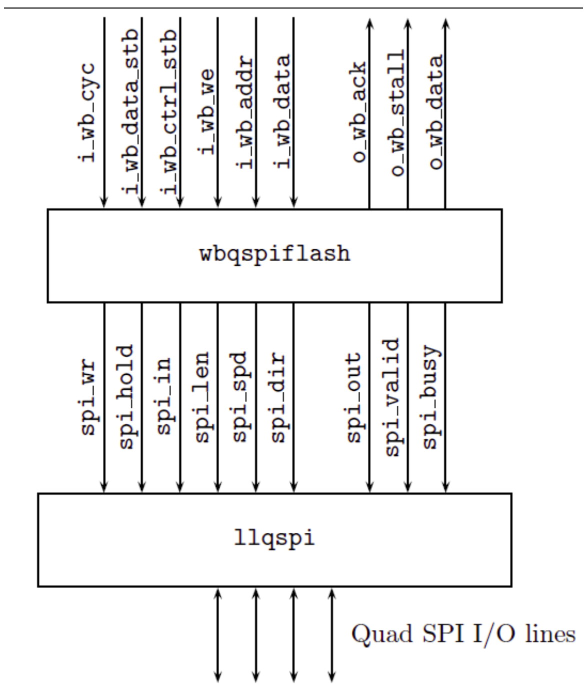


Figure 3.11: QSPI Architecture

high then the **spiBusy** is deasserted for another clock cycle and continue reading the command provided during previous clock cycle. **SpiHold** makes everything to wait until further instructions and halt the clock to the device.The lines is useful in dealing with a slow wishbone bus that is writing to the device but doesnot have its next data ready.Thus by keeping the line **iWbCyc** high, indicates that the the write takes many clock cycle so that the flash can wait for it. The wbqspi flash interacts directly with the wishbone bus.when wbqspiflash is idle then it can handle any command (can be data or control) and can provide necessary commands to the llqspi. For any other states wbqspiflash will halt the bus until it come back to the idle state(there are few exceptions). For subsequent data reads, the device will keep on reading data while for subsequent data writes , if in program mode the device buffer will keep on filling up before the writing starts.In this architecture inorder to reduce complexity, the controlling and issuing of commands are done by higher level component wbqspiflash while the I/Os are connected to lower level driver that will implement actual SPI protocol.

3.4.2 Registers

EREG Register

It is a 32 bit register that will work as a overall control register .Control erase commands and provide status from the device.The last bit will indicate whether a write or erase is in progress .Bit 28 will disable the write protect mode for erase or write operation.The device normally works in SPI mode. The Quad I/O mode can be enabled by setting the corresponding bit in configuration register and bit 27 of EREG register will indicate whether the device is in high speed quad I/O mode.

Configuration register

It is a 8 bit non volatile configuration register where write to this register will stall the bus until the work in progress bit in status or EREG register is cleared. Bit 0 will freeze the BP2-0 bits in status register and these bits are configured using the BPNV(bit 3) of the configuration register.Bit 1 will set the Quad I/O mode.

Status Register

It is a 8 bit register where bit 0 indicate the write in progress and all other bits are set to zero.

3.4.3 Operation

This core will allow the device to read or write via wishbone bus or through the wishbone control register EREG

Erase Operation

- Clear the write protect by writting 0x1000 0000 to the EREG register.
- Erase is commanded by writting 0x8000 0000 plus the address of the sector to be erased to the EREG register.
- Followed by a write enable single from QSPI controller to the device and a sector erase command.
- The controller will enter the idle state and continuously checks the status register, when the device is being erased.
- If a read operation is requested to the EREG at this time,final bit of EREG register will be set indicating that a write is in progress.
- After erasing the sector the write is in progress line will be dropped by status register and a interupt line is activated.
- Attempt to perform any other operation before the completion of erase operation will stall the bus.

Program Operation

- After erasing a sector , it can be programmed.
- Initially, Clear the write protect by writting 0x1000 0000 to the EREG register.
- Write to the erased sector.The first address must start at the beginning of a page to be programmed and must end at the end of the page.

- Writes crossing page boundaries will stall the bus.
- The controller will enter the idle state and continuously checks the status register, when the device is being programmed.
- EREG register and the device status register can be queried during this idle condition.
- When the operation is in progress, the final bit in the EREG register will be set to high indicating the work is in progress .
- During this time attempt to perform any other operation will stall the bus.
- On the completion of the operation,write in progress is deasserted by status register,and the final bit of the EREG register is cleared and interupt lines are activated.

Read Operation

- Request read from the device through the command.
- To read 4 bits at a time from bus ,set or read the quad mode bits in the configuration register.
- When all all 4 I/Os are used,after initial read ,controller will leave the device in special mode where next read starts after a 12 clock setup.
- In bus cycle 20 clocks are required to read the first word and 8 clock per words for next subsequent words.

Instr-type	OPCode	ALUop	Funct7	Funct3	ALUctl	ALUOperation
U	LUI	0000	XXXXXXX	XXX	000100	add
U	AUIPC	0001	XXXXXXX	XXX	000100	add
UJ	JAL	0010	XXXXXXX	XXX	000100	add
I	JALR	0011	XXXXXXX	XXX	000100	add
Branch(S-type)	BEQ	0100	XXXXXXX	000	010100	sub
Branch(S-type)	BNE	0100	XXXXXXX	001	010100	sub
Branch(S-type)	BLT	0100	XXXXXXX	100	000011	lt
Branch(S-type)	BGE	0100	XXXXXXX	101	000011	lt
Branch(S-type)	BLTU	0100	XXXXXXX	110	010011	ltu
Branch(S-type)	BGEU	0100	XXXXXXX	111	010011	ltu
I	LB	0101	XXXXXXX	000	000100	add
I	LH	0101	XXXXXXX	001	000100	add
I	LW	0101	XXXXXXX	010	000100	add
S	SB	1000	XXXXXXX	000	000100	add
S	SH	1000	XXXXXXX	001	000100	add
S	SW	1000	XXXXXXX	010	000100	add
I	ADDI	0111	XXXXXXX	000	000100	add
I	SLTI	0111	XXXXXXX	010	000011	lt
I	SLTUI	0111	XXXXXXX	011	010011	ltu
I	XORI	0111	XXXXXXX	100	000010	xor
I	ORI	0111	XXXXXXX	110	000001	or
I	ANDI	0111	XXXXXXX	111	000000	and
I	SLLI	0111	0000000	001	001000	sll
I	SRLI	0111	0000000	101	011000	srl
I	SRAI	0111	0100000	101	111000	sra
R	ADD	0110	0000000	000	000100	add
R	SUB	0110	0100000	000	010100	sub
R	SLL	0110	0000000	001	001000	sll
R	SLT	0110	0000000	010	000011	lt
R	SLTU	0110	0000000	011	010011	ltu
R	XOR	0110	0000000	100	000010	xor
R	SRL	0110	0000000	101	011000	srl

Instr-type	OPCode	ALUop	Funct7	Funct3	ALUctl	ALUOperation
R	MUL	0110	0000001	000	000101	mul
R	MULH	0110	0000001	001	100101	mulh
R	MULHU	0110	0000001	010	110101	mulhu
R	MULHSU	0110	0000001	011	010101	mulhsu
R	DIV	0110	0000001	100	000110	div
R	DIVU	0110	0000001	101	010110	divu
R	REM	0110	0000001	110	000111	rem
R	REMU	0110	0000001	111	010111	remu

Table 3.3: Intruction decode sequence(contd..)

CHAPTER 4

Software Testing and Simulation

CHAPTER 5

FPGA Testing and Validation

CHAPTER 6

The Way Foreward

CHAPTER 7

Conclusion and Results

CHAPTER 8

Appendix