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

Abstract **i**

List of Figures **iv**

List of Tables **v**

1 Introduction **1**
1.1 Background 1
1.2 Scope 2

2 Design Overview **3**
2.1 Introduction 3
2.2 Constraints 3
2.3 Proposed process 4

3 System Architecture **5**
3.1 Hardware Architecture 5
3.1.1 Core Design 5

4 Detailed Design **7**
4.1 Compute Core 7
4.1.1 Instruction Fetch stage 7
4.1.2 Execution Stage 8
4.2 WISHBONE Bus 12
4.2.1 WISHBONE Basics 13
4.2.2 WISHBONE Features 14
4.3 MIL STD 1553 14
4.3.1 Command Word 16
4.3.2 Status Word 16
4.3.3 Data Word 16

5 Software Testing and Simulation **19**

6 FPGA Testing and Validation **20**

7 The Way Foreward 21

8 Conclusion and Results 22

List of Figures

3.1 Navigation AISC Base Architecture Block Diagram 5

4.1 Instruction Stage Block Schematic 8
4.2 Execution Stage Block Schematic 8
4.3 Wishbone Bus Interface 12
4.4 Interconnection System 13
4.5 MIL STD 1553 15
4.6 Word Formats 16
4.7 Status Word 17
4.8 Command Word 17
4.9 Data Word 18

List of Tables

4.1 ALU Operations with their corresponfing ALUctl line values 9
4.2 Intruction decode sequence(pto) 10
4.3 Intruction decode sequence(contd..) 11

Chapter 1

Introduction

1.1 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. This document is orderedas follows. The Scope and Methodology adopted will be discussed in the following sections. Chapter 2 will discuss the Design overview and how the design process was formulated. Chapter 3 will elaborate on the overall Hardware and Software architecture and will illustrate an Top-Level view of the design. Chapter 4 will be where the design implementation details and detailed architecture will be discussed. All design elements will be elaborated here. Chapter 5 will elaborate how the designed ASIC is to be handled and how the phical chip can be brought onto a printed circuit Board and the necessary configuration and required support peripherals including power and pin mapping. Chapter 6 will conclude this document and shall discuss the overall process that was implemented and the actual results of the project and the road ahead.

1.2 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.

Chapter 2

Design Overview

2.1 Introduction

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.

2.2 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.

2.3 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 3

System Architecture

3.1 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

3.1.1 Core Design

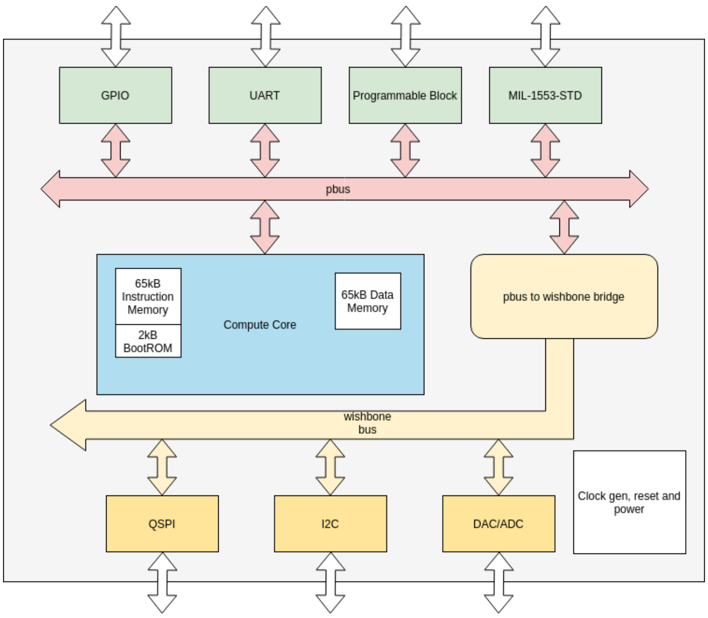

Figure 3.1: Navigation AISC Base Architecture Block Diagram

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),

Chapter 4

Detailed Design

4.1 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.

4.1.1 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.

A block schematic is illustrated in Fig.4.6.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.

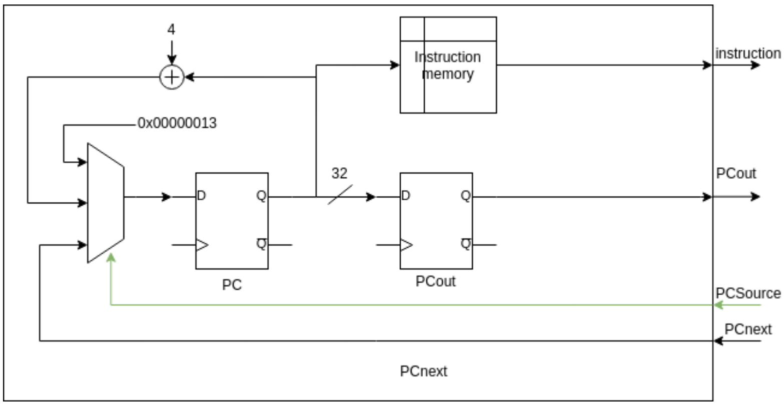

Figure 4.1: Instruction Stage Block Schematic

4.1.2 Execution Stage

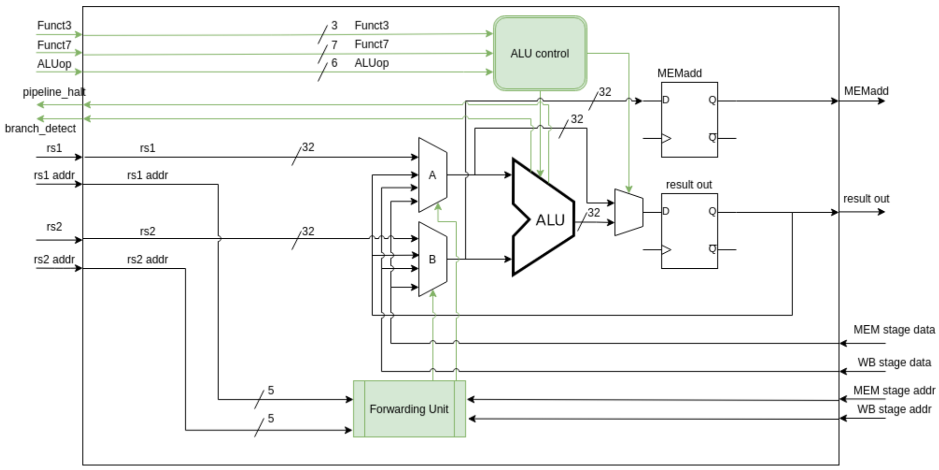

Figure 4.2: Execution Stage Block Schematic

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.4.2.

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 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

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 4.1: ALU Operations with their corresponfing ALUctl line values

of 0101. The operations and their corresponding **ALUctl** values are summarized in Table 4.1

ALUControl

The **ALUctl** values specified in Table 4.1 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 4.2 and 4.3

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 4.1 to 4.6. They are heuristically reduced to a minimum size and implemented

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
R	SRA	0110	0100000	101	111000	sra
R	OR	0110	0000000	110	000001	or
R	AND	0110	0000000	111	000000	and

Table 4.2: Intruction decode sequence(pto)

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

$$\begin{aligned} ALUctl[0] = & a_3'a_2a_1'a_0'b_2 + a_3'a_2a_1a_0b_2'b_1 + a_3'a_2a_1a_0b_2b_1b_0' + \\ & a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0'b_2'b_1 \\ & + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0'b_2b_1b_0' \\ & + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0b_2'b_1' + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0b_1 \end{aligned} \tag{4.1}$$

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 4.3: Intruction decode sequence(contd..)

$$\begin{aligned} ALUctl[1] =& a_3'a_2a_1'a_0'b_2 + a_3'a_2a_1a_0b_2'b_1+ \\ & a_3'a_2a_1a_0b_2b_1'b_0' + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0'b_2'b_1+ \\ & a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0'b_2b_1'b_0' + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0b_2 \end{aligned} \tag{4.2}$$

$$\begin{aligned} ALUctl[2] =& a_3'a_2' + a_3'a_2a_1b_2'b_1' + a_3'a_2a_1'a_0b_2'b_1b_0' + a_3'a_2a_1'a_0b_2b_1' \\ & + a_3a_2'a_1'a_0'b_2'b_1' + a_3a_2'a_1'a_0'b_2'b_1b_0' + a_3'a_2a_1a_0b_2'b_1'b_0' \\ & + a_3'a_2a_1a_0'f_6f_4'f_3'f_2'f_1'f_0'b_2'b_1'b_0' + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0 \end{aligned} \tag{4.3}$$

$$ALUctl[3] = a_3'a_2a_1f_6'f_5'f_4'f_3'f_2'f_1'f_0'b_2'b_1'b_0 + a_3'a_2a_1f_6'f_4'f_3'f_2'f_1'f_0'b_2b_1'b_0 \tag{4.4}$$

$$\begin{aligned} ALUctl[4] =& a_3'a_2a_1'a_0'b_2'b_1' + a_3'a_2a_1'a_0'b_2b_1 + a_3'a_2a_1a_0b_2b_1b_0+ \\ & a_3'a_2a_1a_0f_6'f_4'f_3'f_2'f_1'f_0'b_2b_1'b_0 + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0'b_2'b_1'b_0'+ \\ & a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0'b_2'b_1b_0 + a_3'a_2a_1a_0'f_6'f_4'f_3'f_2'f_1'f_0'b_2b_1'b_0'+ \\ & a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0b_2'b_1 + a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0b_2b_0 \end{aligned} \tag{4.5}$$

$$ALUctl[5] = a_3'a_2a_1a_0'f_6'f_5'f_4'f_3'f_2'f_1'f_0b_2'b_0 + a_3'a_2a_1f_6'f_5f_4'f_3'f_2'f_1'f_0'b_2b_1'b_0 \tag{4.6}$$

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)

4.2 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. With use of this standardize bus interface it is much easier to connect the cores, and therefore much easier to create a custom System-on-Chip.

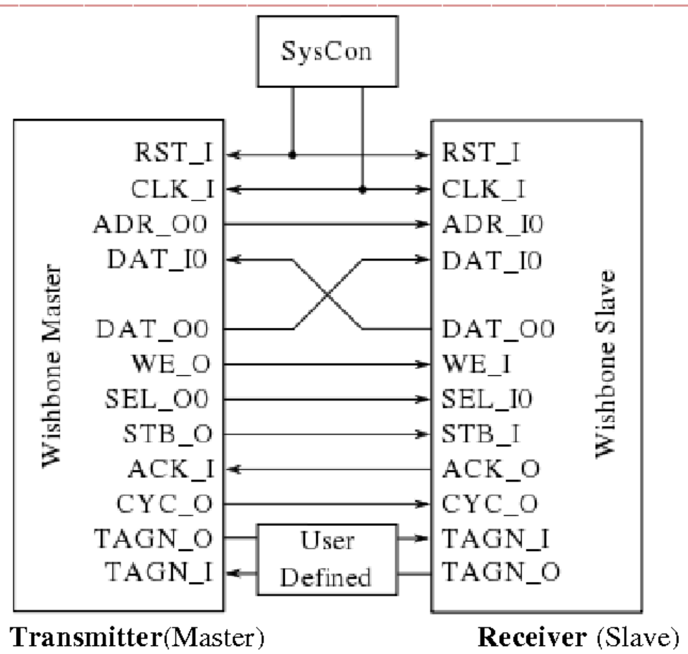

Figure 4.3: Wishbone Bus Interface

This way of SOC design improves the portability and reliability of the system, and results in faster time-to-market for the end user. The objective behind WISHBONE is to create a portable interface that supports both FPGA and ASIC that is independent of the semiconductor technology and WISHBONE interfaces should be independent of logic signaling levels.

Another important reason is to create a flexible interconnection scheme that is independent of the type of IP core delivery (Hard, Soft IP) method. The next reasons are to have a standard interface that can be written using any hardware description language such as VHDL and VERILOG. It supports a variety of bus transfer cycle in which the data transaction is independent of the application specific functions of the IP cores. It also supports different types of interconnection architectures with theoretically infinite range of operating frequency. The final objective of WISHBONE bus is that it is absolutely free to use by developers without paying any fee for the cores available.

4.2.1 WISHBONE Basics

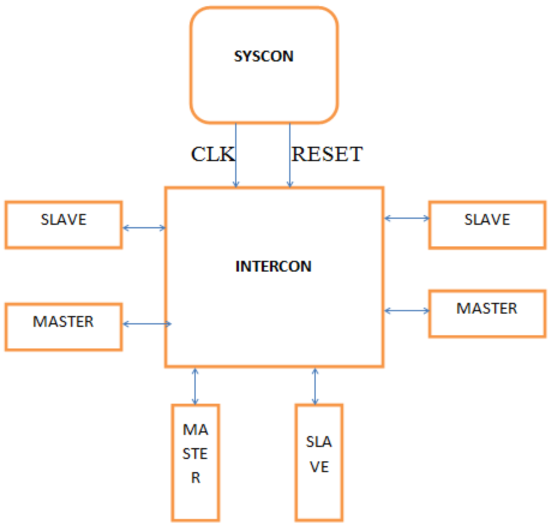

Figure 4.4: Interconnection System

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.Master starts transaction providing an address and control signal to Slave. Slave in turn 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. The Intercon also requires a “SYSCON” module which generates WISHBONE reset and clock signal for the proper functioning of the system. Figure:4.4 show the WISHBONE Intercon system which consists of Masters and Slaves and SYSCON modules. WISHBONE Intercon can be designed to operate over an infinite frequency range. This is called as variable time specification. The speed of the operation is only limited by the technology of the integrated circuits. The interconnection can be described using hardware description languages like VHDL and Verilog, and the system integrator can modify the interconnection according to the requirement of the design. Hence WISHBONE interface is different from traditional microcomputer buses such as PCI, VME bus and ISA bus.

4.2.2 WISHBONE Features

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

1. Simple, compact, logical IP core hardware interfaces that require very few logic gates.
2. Variable core interconnection methods support point-to-point, shared bus, crossbar switch, and switched fabric interconnections.
3. Handshaking protocol allows each IP core to throttle its data transfer speed.
4. Supports single clock data transfers.
5. MASTER / SLAVE architecture for very flexible system designs.
6. Synchronous design assures portability, simplicity and ease of use.
7. 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 (bigger) address. In the second way, the most significant byte of an operand can be placed at the lower (smaller) address. These are called BIG ENDIAN and LITTLE ENDIAN data operands, respectively. WISHBONE supports both types.

4.3 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)

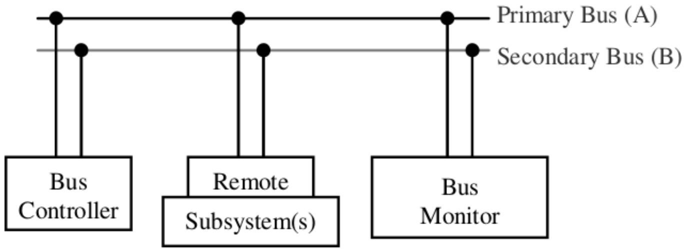

Figure 4.5: MIL STD 1553

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 with the secondary bus 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 theinformation transfers through the data bus. The standard specifies the informationtransfers between RTs to follow a command/response format.The BC sendscommands 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 RT can be an independent subsystem or it can be portion of the subsystem. The Standard allows for up to 31 RTs in a system.The MT 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. Three bits out of 20 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.

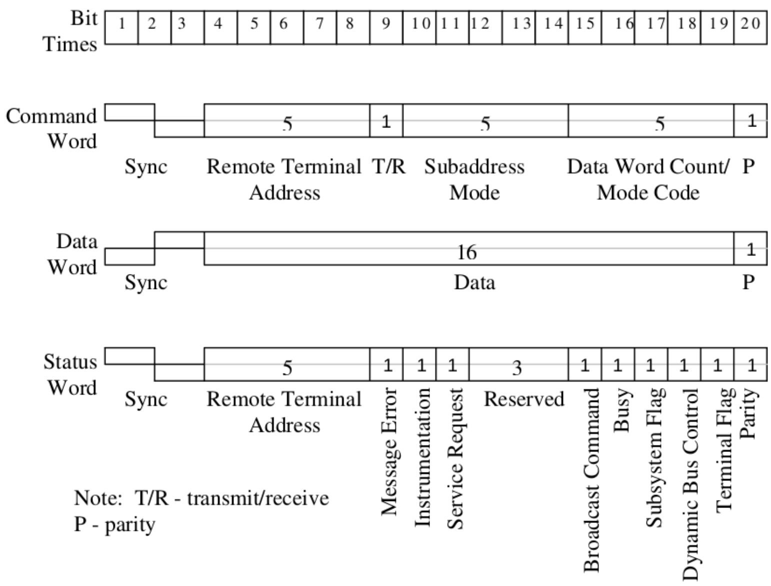

Figure 4.6: Word Formats

4.3.1 Command Word

The BC issues "Command Word (CW)" to RTs to perform specific functions. The CW is shown in the Figure 4.8.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.

4.3.2 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 4.7

4.3.3 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 4.9.

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	Instrumentat ion	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 4.7: Status Word

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 4.8: Command Word

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 4.9: Data Word

Chapter 5

Software Testing and Simulation

Chapter 6

FPGA Testing and Validation

Chapter 7

The Way Foreward

Chapter 8

Conclusion and Results