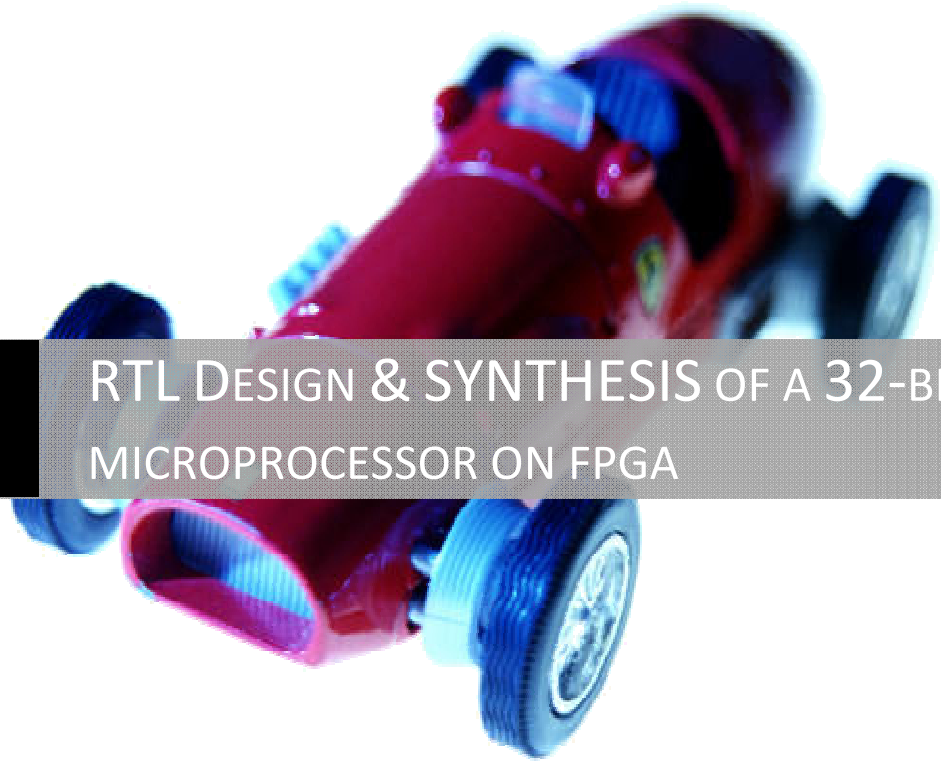
5/5/2011



SRM UNIVERSITY

RTL DESIGN & SYNTHESIS OF A 32-BIT MICROPROCESSOR ON FPGA

Submitted in the partial fulfillment for the award of B. Tech. Degree

RTL DESIGN & SYNTHESIS

OF

A 32-BIT MICROPROCESSOR ON FPGA

Submitted in partial fulfillment of the requirement for the award of the degree of

B. Tech.

In

Electronics & Communication Engineering

Submitted By

Deepak Kumar Gupta(10407837)

Anand Kumar Singh(10407820)

Gautam Kr. Sinha(10407844)

Anindita Panda(10407823)

Under the Supervision of

Mr. Prashant Mani

(Asst. Prof.)

Electronics & Communication Dept.

SRM University, Ghaziabad

Department of Electronics & Communication Engineering

SRM University, NCR Campus,Modinagar

Ghaziabad-201204

May-2011

Bonafide Certificate

This is to certify that the project work entitled **"RTL DESIGN & SYNTHESIS OF A 32-BIT MICROPROCESSOR ON FPGA"** has been successfully completed by Deepak Kumar Gupta(10407837), Anand Kumar Singh(10407820), Gautam Kumar Sinha(10407844) & Anindita Panda(10407823) under my supervision. Their work was found to be diligent & I wish them all success for their future.

Project Guide

Mr. Prashant Mani

(Asst. Prof., SRM University)

Project Coordinators

Mr. Pankaj Singh	**Mr.Debasis Haldar**	**Mr. Pankaj Agarwall**
(Asst. Prof., SRM University)	(Asst. Prof., SRM University)	(Asst. Prof., SRM University)

Dr. Prof. Manoj Pandey

HOD

Internal Examiner **External Examiner**

Department of Electronics & Communication Engineering

SRM University, NCR Campus, Modinagar

Ghaziabad-201204

Declaration

We do hereby declare that the project report entitled **"RTL DESIGN & SYNTHESIS OF A 32-BIT MICROPROCESSOR"** is a record of original work carried out by us under the supervision of Mr. Prashant Mani (Asst. Prof., Dept. of Electronics & Communication Engineering).

This project work has not been submitted earlier in part or full for the award of any degree or diploma.

Deepak Kumar Gupta Anand Kumar Singh Gautam Kr. Sinha Anindita Panda

Place: Modinagar

Date: …./05/2011

Acknowledgement

We are highly thankful to Mr. Prashant Mani for his relentless support & guidance. We are also thankful to our Project coordinators- Mr. Pankaj Singh, Mr. Haldar & Mr. Pankaj Agarwall for putting their faith upon us & allowing us to continue such a challenging project.

We are also highly obliged to Mr. Aditya Agarwall who guided us to our next step, i.e. FPGA Dumping. Also we would not forget the valuable contribution made by Mr. Ramesh & Mr. Pramod who were always with us whenever we required their assistance.

Also we are thankful to each other, as each made significant contribution by following the right strategy, & maintaining patience even in tough times.

Not to forget the contribution of our H'nble dean sir, Dr. Prof. Manoj Pandey who at the final moment provided us with his precious time suggesting several improvements in our design.

Last but not the least we would like to thank Mr. Sanjeev Rai, (Prof., Electronics & Communication Engineering, and MNNIT Allahabad) for putting his interest in our project and providing such a huge wave of encouragement in our work.

Once again we wish to acknowledge our gratitude to all those people who were involved either directly or indirectly for the competition of this task.

With Warm Regards

Deepak Kumar Gupta

Anand Kumar Singh

Gautam Kr. Sinha

Anindita Panda

Abstract

This is a very challenging project because processors are not as flexible as programmable logic. The ability to emulate a microprocessor on a programmable chip can lead to cheaper, more efficient and more flexible performance. But then today there is also a second stringent demand that is miniaturization with minimum power delay product. The race is on for reaching the minimum space as possible occupied by the systems.

Our project, keeping an eye wide open on the future will mix both fields enabling us to communicate, in a digital manner, by using systems that would be integrated on an IC through programming of the IC using VHDL**.**

Our goal is to design a 32-bit microprocessor in VHDL. All of the functionality will come from the codes written in VHDL. The processor will have all of the arithmetic and logic functions that are on a standard 32-bit microprocessor. To give the project some user interaction, we set up the instruction input to be sent in one instruction at a time via a set of dipswitches and a pushbutton on the FPGA board. Future work would include the implementation of JUMP instructions, interrupts, and adding more addressing modes.

Table of Contents

Chapter 1

Introduction

1.1 Overview

The development of a bipolar technology for integrated circuits (ICs) went hand in hand with the steady improvement in semiconductor materials and discrete components during the 1950s and 1960s. Consequently, silicon bipolar technology formed the basis for the IC market during the 1970s. As circuit dimensions shrink, the MOSFET (or MOS) has gradually taken over as the major technological platform for silicon ICs. The main reasons are the ease of miniaturization and high yield for MOS compared with bipolar technology. For VLSI circuits the low standby power of complementary MOS (CMOS) gates is a significant advantage compared with integrated bipolar circuits.

According to the famous theory stated by Sir Moore's "the number of transistors that can be inexpensively placed over an integrated circuit doubles itself every 2 years" the capabilities of IC has increased exponentially over the years, in terms of computation power, utilization of available area, yield. The combined effect of these two advances is that people can now put diverse functionality into the IC's, opening up new frontiers. Examples are embedded systems, where intelligent devices are put inside everyday objects, and ubiquitous computing where small computing devices proliferate to such an extent that even the shoes we wear may actually do something useful like monitoring our heartbeats!!

Graph for Moore's Law

A microprocessor is a multipurpose, programmable clock driven, register based electronic device manufactured by using either a Large Scale Integration (LSI) or Very Large Scale Integration (VLSI) technique that reads binary instruction from a storage device called memory except binary data as input and process data calling to those instruction and provide result as output. The physical components of this system are called as hardware. Similarly a set of instructions written for the microprocessor to perform a task is called a program, and a group of programs is called software.

Basically a microprocessor controls 3 main tasks for the computer system:

- Data transfer between itself and the memory or input output systems.
- Arithmetic and Logic Operations and
- Program flow via simple decisions.

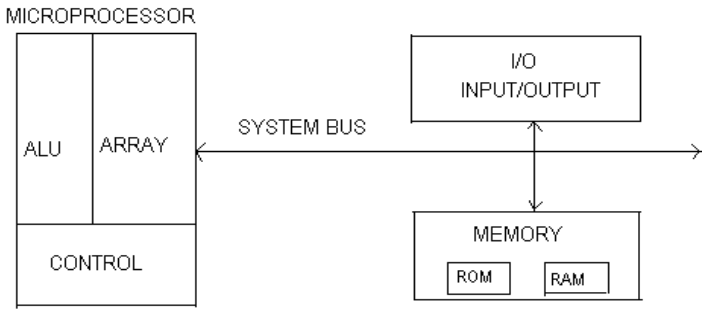
1.2 Problem Statement

Aim	**:**	**RTL Design & Synthesis of a 32-bit Microprocessor**
Tools Used	**:**	**Xilinx ise9.1**
		Modelsim SE5.7f
Hardware Used	**:**	**Spartan 3E**
Product Category	**:**	**General**
Family	**:**	**Spartan 3E**
Device	**:**	**XC3S500E**
Package	**:**	**FG320**
Speed	**:**	**-5**
Top Level Source Type:		**HDL**
Synthesis Tool	**:**	**XST (VHDL/VERILOG)**
Simulator	**:**	**Modelsim-SE VHDL**
Preferred Language	**:**	**VHDL**

Chapter 2

LITERATURE SURVEY

2.1 Microprocessor Architecture



Block diagram of a Microprocessor

The processor contains a number of basic pieces. There is a register array of eight 32-bit registers, an ALU (Arithmetic Logic Unit), a shifter, a program counter, an instruction register, a comparator, an address register, and a control unit. All of these units communicate through a common, 32-bit tristate data bus.

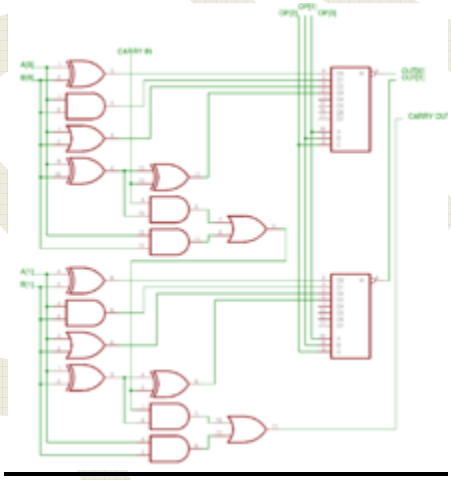
2.1.1 Arithmetic & Control Unit

In computing, an arithmetic logic unit (ALU) is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit (CPU) of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern CPUs and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs.

Mathematician John von Neumann proposed the ALU concept in 1945, when he wrote a report on the foundations for a new computer called the EDVAC. Research into ALUs remains an important part of computer science, falling under Arithmetic and logic structures in the ACM Computing Classification System.

Most ALUs can perform the following operations:

- Bitwise logic operations (AND, NOT, OR, XOR)
- Integer arithmetic operations (addition, subtraction, and sometimes multiplication and division, though this is more expensive)
- Bit-shifting operations (shifting or rotating a word by a specified number of bits to the left or right, with or without sign extension). Shifts can be seen as multiplications and divisions by a power of two.



A simple example arithmetic logic unit (2-bit ALU) that does AND, OR, XOR, and addition

2.1.2 Control Unit

The Control Unit provides control signals necessary to control the hardware of the CPU & may be hardwired where signals are generated by a combinational logic circuit in a fashion much faster. It is less flexible & is harder to design and debug.

It can even be micro programmed and signals are stored in control memory thus as a result it is slower than hardwired yet more flexible and easier to design and debug where control signals are

needed to control functions of various hardware units and to direct the flow of information within the CPU.

2.1.3 Memory

It is used to store programs and data which is volatile and usually uses DRAM…dynamic random access memory and is slower than static ram as it needs to be refreshed.

Apart from that it requires fewer transistors to implement and thus improves packing density on IC…allowing larger, cheaper memories where most memory is byte addressable. It retrieves a single byte per memory access and can be organized to access a full word or even multiple words per access with its cache memory is a distinct memory positioned between the CPU and MM resulting in faster operation, smaller size with the tradeoff of between performance and cost.

2.1.4 Input Output Devices

In computing, **input/output**, or **I/O**, refers to the communication between an information processing system (such as a computer), and the outside world, possibly a human, or another information processing system. Inputs are the signals or data received by the system, and outputs are the signals or data sent from it. The term can also be used as part of an action; to "perform I/O" is to perform an input or output operation. I/O devices are used by a person (or other system) to communicate with a computer. For instance, a keyboard or a mouse may be an input device for a computer, while monitors and printers are considered output devices for a computer. Devices for communication between computers, such as modems and network cards, typically serve for both input and output.

2.1.5 Register Array

It mainly comprises of the following types:

1. Instruction Register
2. Memory Address register
3. Memory Buffer Register

2.1.5.1 Instruction Register

The instruction register contains the currently executing instruction and holds the instruction while it is being decoded while the opcode field provides input to the control system indicating the operation to be performed. It contains addresses of the operands to be used in operation and contains the destination address of the result and also contains information about the addressing modes to be used.

2.1.5.2 Memory Address Register

It holds the address of the location in memory to be accessed and this may be the address of the next instruction to be fetched or may be the address of an operand to be read from memory and can even be the address of information to be written to memory.

2.1.5.3 Memory Buffer Register

It holds values to be transferred between the main memory and the CPU and reads the data or instructions from the memory and writes the values to the memory. Most of the machines are capable of transferring more than a single word.

2.2 Operations of processor

The functionality of a microprocessor is based on 3-states cycle:

- Fetch– Bring the next instruction to the processor.
- Decode– Determine what is required and what to do.
- Execute– Operate the instruction.

2.3 Design

A microprocessor is a wonderful piece of example of the VLSI design world with several thousands of transistors devised within a single IC chip.

VLSI stands for "Very Large Scale Integration". This is the field which involves packing more and more logic devices into smaller and smaller areas. Thanks to VLSI, circuits that would have taken huge space can now be put into a small space few millimeters across! This has opened up a big opportunity to do things that were not possible before. VLSI circuits are everywhere ... computer, cars, state-of-the-art digital camera, and the cell-phones. All this involves a lot of

expertise on many fronts within the same fields.

2.3.1 THE VLSI DESIGN PROCESS

A typical digital design flow is as follows:

- Specification
- Architecture
- RTL Coding
- RTL Verification
- Synthesis
- Backend
- Tape Out to Foundry to get end product….a wafer with repeated number of identical IC's.

All modern digital designs start with a designer writing a hardware description of the IC (using HDL or Hardware Description Language) in Verilog/VHDL. A Verilog or VHDL program essentially describes the hardware (logic gates, Flip-Flops, counters etc) and the interconnects of the circuit blocks and the functionality.

This language is used to design the circuits at a high-level, in two ways. It can either be a behavioral description, which describes what the circuit is supposed to do, or a structural description, which describes what the circuit is made of. There are other languages for describing circuits, such as Verilog, ABEL which work in a similar fashion.

A typical analog design flow is as follows:

- Specifications
- Architecture
- Circuit Design
- SPICE Simulation
- Layout
- Parametric Extraction / Back Annotation

- Final Design
- Tape Out to foundry.

While digital design is highly automated now, very small portion of analog design can be automated. There is a hardware description language called AHDL but is not widely used as it does not accurately give us the behavioral model of the circuit because of the complexity of the effects of parasitic on the analog behavior of the circuit. Many analog chips are what are termed as “flat” or non-hierarchical designs. Most of today’s VLSI designs are classified into the following 3 categories:

1. **Analog:**

Small transistor count precision circuits such as Amplifiers, Data converters, filters, Phase Locked Loops, Sensors etc.

2. **ASICS or Application Specific Integrated Circuits:**

Progress in the fabrication of IC's has enabled us to create fast and powerful circuits in smaller and smaller devices. This also means that we can pack a lot more of functionality into the same area. The biggest application of this ability is found in the design of ASIC's. These are IC's that are created for specific purposes - each device is created to do a particular job, and do it well. The most common application area for this is DSP - signal filters, image compression, etc. To go to extremes, consider the fact that the digital wristwatch normally consists of a single IC doing all the time-keeping jobs as well as extra features like games, calendar, etc.

3. **SoC. or Systems on a chip:**

These are highly complex mixed signal circuits (digital and analog all on the same chip). A network processor chip or a wireless radio chip is an example of a SoC.

2. 4 Introduction to VHDL

VHDL is a hardware description language. It describes the behavior of an electronic Circuit or system, from which the physical circuit or system can then be attained or Implemented.VHDL

stands for VHSIC Hardware Description Language. VHSIC is itself an abbreviation for Very High Speed Integrated Circuits, an initiative funded by the United States Department of Defense in the 1980s that led to the creation of VHDL. Its first version was VHDL 87, later upgraded to the so-called VHDL 93. VHDL was the original and first hardware description language to be standardized by the Institute of Electrical and Electronics Engineers, through the IEEE 1076 standard. An additional standard, the IEEE 1164, was later added to introduce a multi-valued logic system.

2.4.1 Advantages of Using VHDL

- VHDL can be used as an exchange medium between chip vendors and CAD tool users.
- VHDL can be used as an exchange medium between different CAD and CAE tools.
- It supports hierarchy.
- It supports flexible design methodology.
- Even though being non technology specific it supports technology specific features.
- Supports both synchronous and asynchronous timing models.
- Supports wide range of abstraction levels.

Chapter 3

System Specification

3.1 Terms used in Microprocessors

BIT	: A digit of the binary number or code is called a bit.
NIBBLE	: The 4-bit binary number or code is called a nibble.
BYTE	: An 8 bit binary number or code is called a byte.
WORD	: The 16 bit binary number or code is called a word.
DOUBLE WORD	: A group of 32 bit binary number or code is called a double word.
DATA	: The quantity (binary number /code) operating by an instruction of program is called data.
ADDRESS	: The address is an identification number in binary form for memory locations.
MEMORY SIZE	: It is the size of binary information that can be stored in the memory location.
BUS	: A bus is a group of conducting lines or wires that carry data, address and control signal. Buses can be classified into data bus, address bus and control bus.
ADDRESS BUS	: Groups of conducting lines or wires that carry address only are called as address bus.
DATA BUS	: Groups of conducting lines or wires that carry data only are called as data bus.
CONTOL BUS	: Groups of conducting lines or wires that carry control signals only are called as control bus.
CPU BUS	: Groups of conducting lines that are directly connected to a microprocessor are called CPU buses. In CPU bus the signal are multiplexed that is more than one signal are passed through the same line but at different timings.

SYSTEM BUS : The groups of conducting lines that carry data, address, and control signals in a micro computer system are called system bus. Multiplexing is not allowed in system buses.

CLOCK : A clock is a square wave generator, which is used to synchronize various devices in the microprocessor and in the system. Every microprocessor system requires a clock for its functioning. The time taken by the microprocessor and the system to execute an instruction or program is measured only in terms of the time period of its clock.

DUTY CYCLE : In engineering, the **duty cycle** of a machine or system is the time that it spends in an active state as a fraction of the total time under consideration

3.2 Evolution of Microprocessors

In 1971, Intel Corporation released the world's first microprocessor the INTEL 4004, a 4-bit microprocessor. It addresses 4096 memory locations of word size 4-bit.the instruction set consists of 45 different instructions. It is monolithic IC employing large scale integration in PMOS technology. The INTEL 4004 was soon followed by a variety of microprocessors, with the most of the major semiconductor manufactures one or more types.

3.3 Generation of Microprocessors

3.3.1 First Generation Microprocessors

The microprocessors introduced between 1971 and 1973 were the first generation processors. They were designed using PMOS technology. This technology provided low cost, slow speed and low output currents and was not compatible with TTL (Transistor transistor logic) levels. The first generation processors required a lot of additional support IC's to form a system. It required as high as 30 IC's to form a system. The 4-bit processors were provided with only 16 pins and 8 bit& 16 bit processors were provided with 40 pins. Due to limitations of pins the signal was multiplexed. A list of first generation microprocessors are given below:

3.3.2 Second Generation Microprocessors

By the late 1970s (specifically 1973), enough transistors were available on the IC to usher in the second generation of microprocessor sophistication: 16-bit arithmetic and pipelined instruction processing. Motorola's MC68000 microprocessor, introduced in 1979, is an example. Another example is Intel's 8080. This generation is defined by overlapped fetch, decode, and execute steps (Computer 1996). As the first instruction is processed in the execution unit, the second instruction is decoded and the third instruction is fetched. The distinction between the first and second generation devices was primarily the use of newer semiconductor technology to fabricate the chips. This new technology resulted in a five-fold increase in instruction, execution, speed, and higher chip densities.

3.3.3 Third Generation Microprocessors

The third generation, introduced in 1978, was represented by Intel's 8086 and the Zilog Z8000, which were 16-bit processors with minicomputer-like performance. The third generation came about as IC transistor counts approached 250,000. Motorola's MC68020, for example, incorporated an on-chip cache for the first time and the depth of the pipeline increased to five or more stages. This generation of microprocessors was different from the previous ones in that all major workstation manufacturers began developing their own RISC-based microprocessor.

3.3.4 Fourth Generation Microprocessors

As the workstation companies converted from commercial microprocessors to in-house designs, microprocessors entered their fourth generation with designs surpassing a million transistors. Leading-edge microprocessors such as Intel's 80960CA and Motorola's 88100 could issue and retire more than one instruction per clock cycle (Computer, 1996).

3.3.5 Fifth Generation Microprocessors

Microprocessors in their fifth generation, employed decoupled super scalar processing, and their design soon surpassed 10 million transistors. In this generation, PCs are a low-margin, high-volume-business dominated by a single microprocessor (Computer, 1996).

3.4 Companies associated with microprocessors

Overall, Intel Corporation dominated the microprocessor area even though other companies like Texas Instruments, Motorola, etc also introduced some microprocessors.

Chapter 4

System Architecture

The processor contains a number of basic pieces there is a register array of eight 32-bit register, an ALU (Arithmetic logic unit), a shifter, a program counter, an instruction register, a comparator, an address register and a control unit. All of these units communicate through a common, 32-bit tristate data bus.

4.1 Components

The microprocessor design has been split into eleven different units or modules, where each module has been separately designed using structural type of modeling and then finally combining them using behavioral modeling.

- Cpu Library
- Cpu Block
- Arithmetic & Logical Operations Block
- Comparator
- Control Unit Block
- Memory
- Shifter Block
- Register Block
- Register Array Block
- Triregister Block
- Top Level Module

The coding for each section has been described in detail in the appendix section.

4.1.1 Cpu Library

A top-level package **cpu_lib.vhd** is needed to describe the signal types that are used to communicate between the CPU components. This package describes a number of types that are used to specify the **ALU** functionality, the shifter operation, and the states needed for the control of the **CPU**.

4.1.2 Cpu Block

Architecture **rtl** of entity **CPU** is a structural implementation of the block diagram. Architecture **rtl** contains the component declarations of all of the components used to build the design, the signals used to connect the components, and the component instantiations to create the functionality. After the component and signal declarations are the component instantiation statements that instance the components and connect the appropriate signals.

4.1.3 Arithmetic & Logical Operations Block

The first entity described is the ALU. This entity performs a number of arithmetic or logical operations on one or more input busses. A symbol for the ALU is shown in figure 3.1

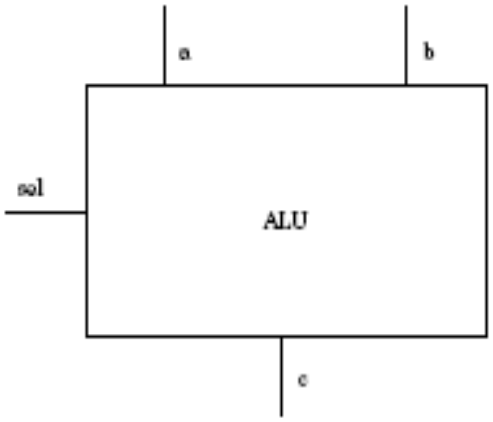


Figure3.1

4.1.4 Comparator

The next component described is the comparator entity **comp**. This entity compares two values and returns either a **'1'** or **'0'** depending on the type of comparison requested and the values being compared. A symbol showing the ports of the comparator is shown in Figure3.2

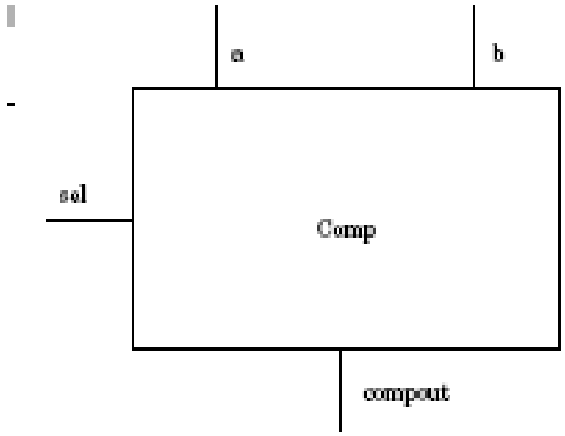


Figure 3.2

The comparison type is determined by the value on input port **sel**. For instance, to compare if inputs **a** and **b** are equal, apply the value **eq** to port **sel**. If ports **a** and **b** have the same value, port **compout** returns **'1'**. If the values are not equal, **'0'** is returned. The types of comparisons allowed are described by type **t_comp** in package **cpu_lib** in file **cpulib.vhd** described earlier. The full table of comparison types and values is shown in table3.2

All operations work on two input values and return a single bit result. This bit is used to control the flow of operation within the processor while executing instructions.

4.1.5 Control Unit

The **control** entity provides the necessary signal interactions to make the data flow properly through the CPU and perform the expected functions. Architecture **rtl** contains a state machine that causes all appropriate signal values to update based on the current state and input signals and produce a next state for the state machine.

The control symbol has only a few inputs, but a lot of outputs. The control block provides all of the control signals to regulate data traffic for the CPU.

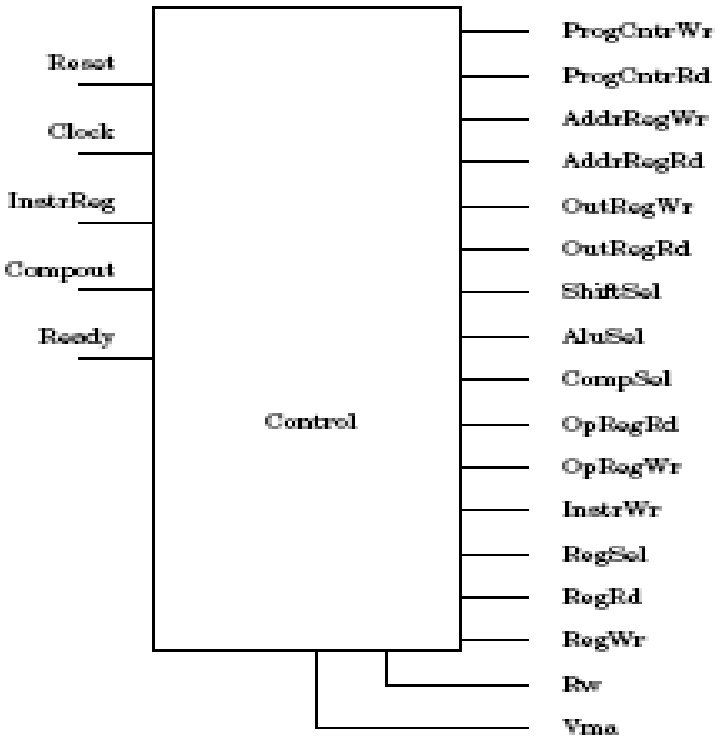


Figure 3.10

4.1.6 Memory

Entity **mem** is a large array with a simple bus interface to allow reading and writing to the memory. A memory location is selected for **read** by placing the appropriate address of the location on signal **addr**, setting input **rw** (read write) to a **'0'** and putting the value **'1'** on signal **sel** (select). The value of the memory location appears on signal **data**, and signal **ready** is set to a **'1'** value signaling that the memory information is available.

4.1.7 Shift Unit

The next device to be described is the **shift** entity. The **shift** entity is used to perform shifting and rotation operations within the CPU. The **shift** entity has a 32-bit input bus, a 32-bit output bus, and a **sel** input that determines which shift operation to perform. This is shown by the symbol in Figure 3.3.

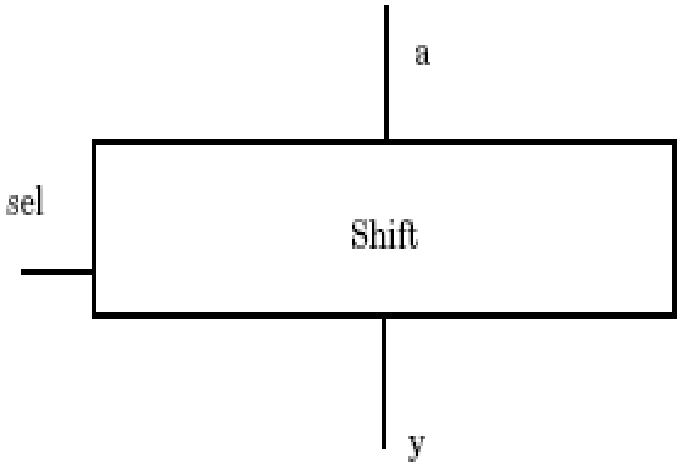


Figure 3.3

This unit performs the following functions:

- Shift logical left
- Shift logical right
- Shift left arithmetic
- Shift right arithmetic
- Shift rotate left
- Shift rotate right

4.1.8 Register

The **reg** entity is used for the address register and the instruction register. These registers need to be able to capture the input data on a rising edge of the **clk** input and drive output **q** with the captured data. The value of input **a** is assigned to output **q** when a rising edge occurs on input **clk**. The assignment is delayed by 1 nanosecond to remove delta delay problems during simulation. A symbol for the **reg** entity is shown in Figure 3.8

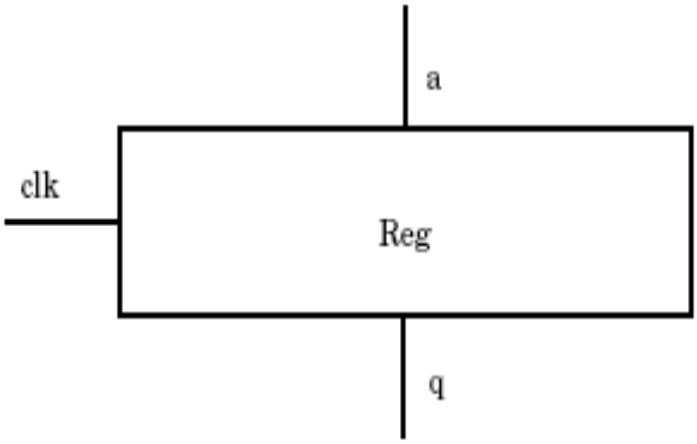


Figure 3.8

The **reg** symbol contains three ports. Port **a** is the data input port; port **q** is the data output port, and port **clk** controls when the data is stored in the **reg** entity. Following is the VHDL description for entity **reg**.

4.1.9 Register Array

The **regarray** entity is used to model the set of registers within the CPU that are used to store intermediate values during instruction processing. These registers are read from and written to during the execution of instructions. The set of registers is modeled as a RAM of eight 16-bit words. The symbol for the **regarray** entity is shown in Figure 3.11.

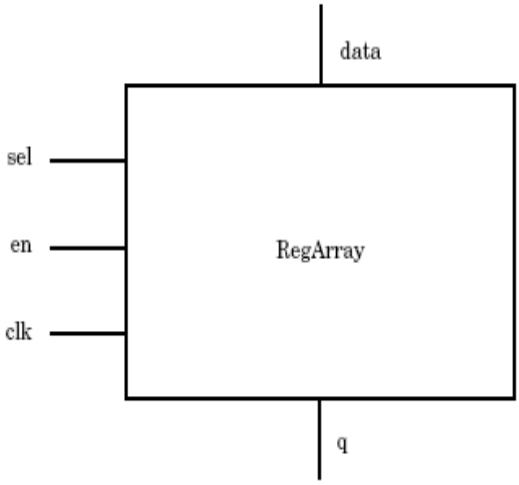


Figure 3.11

4.1.10 Triregister

The next component of the CPU is the tristate register component, **trireg**. The tristate register is connected to the main data bus and can store information from the data bus as well as drive information to the data bus. The **trireg** entity has four ports as shown in Figure 3.9. Input **a** is the data input to the register, and port **q** is the data output from the register. Input **clk** is used to store a new value into the register.

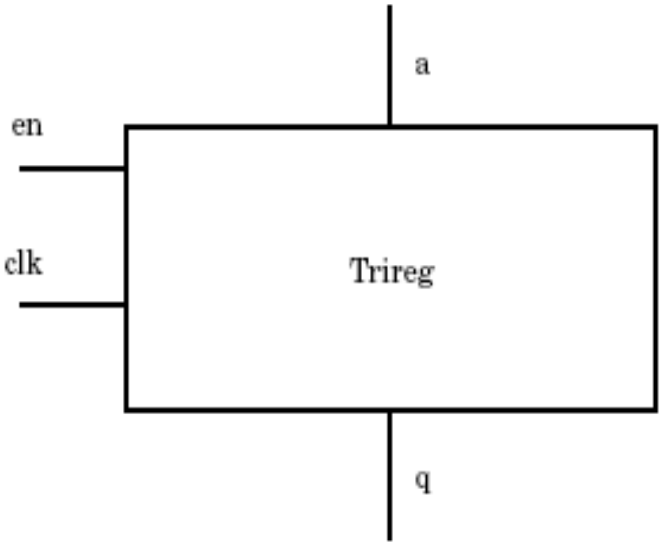


Figure 3.9

4.1.11 Top Level System Design

First of all, a top-level package **cpu_lib.vhd** is needed to describe the signal types that are used to communicate between the CPU components.

This model instantiates components **CPU** and **mem** and specifies the necessary signals to connect the components, as shown in Figure 3.15. Component **mem** is a memory device and contains the instructions and data for the CPU to execute. Component **cpu** is an RTL implementation of the CPU device that is simulated for correctness and synthesized to implement the design.

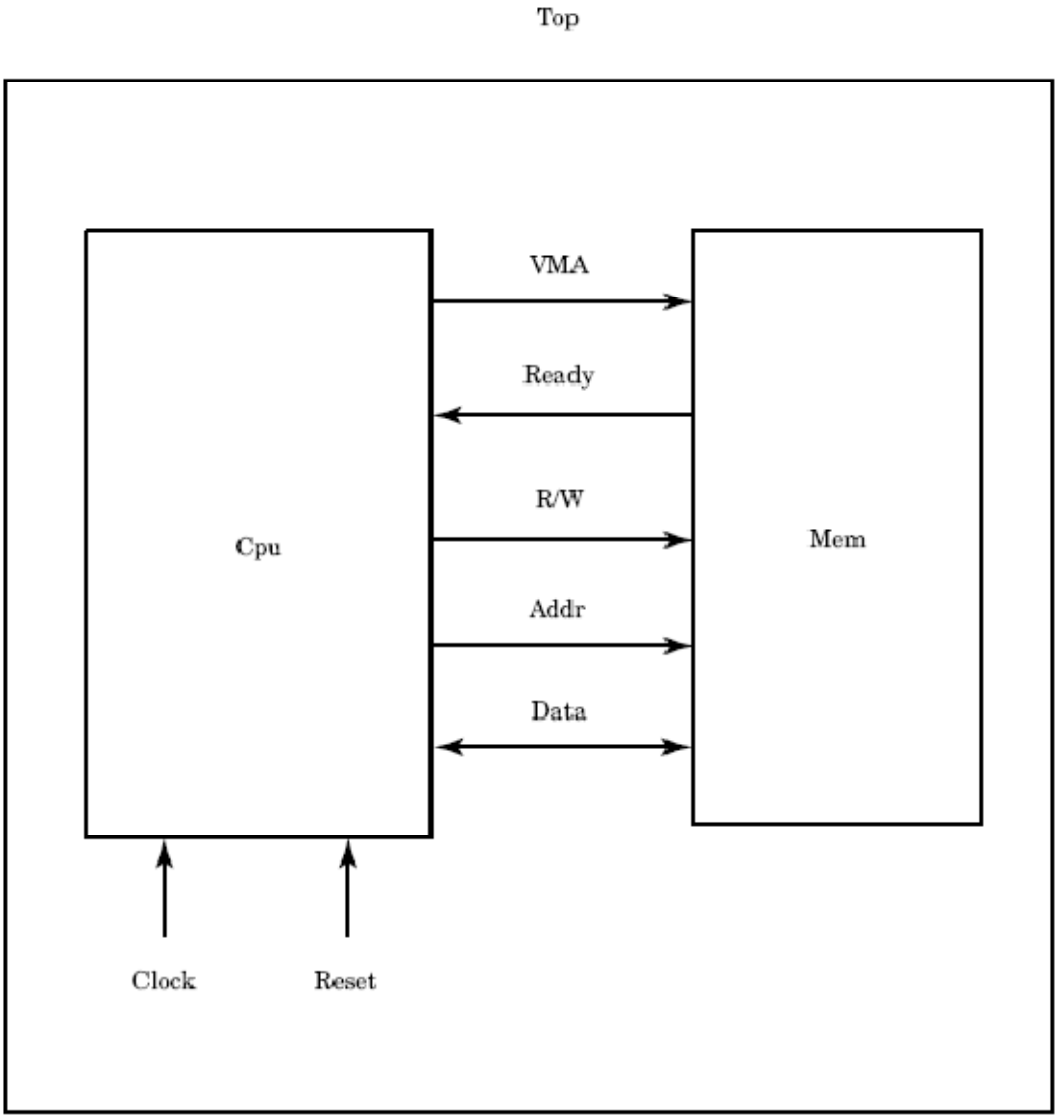


Figure 3.15

5. FPGA (Field Programmable Gate Array):

Abbreviations used in this context

IOB=Input/output block

CLB=Configurable Logic block (static ram based)

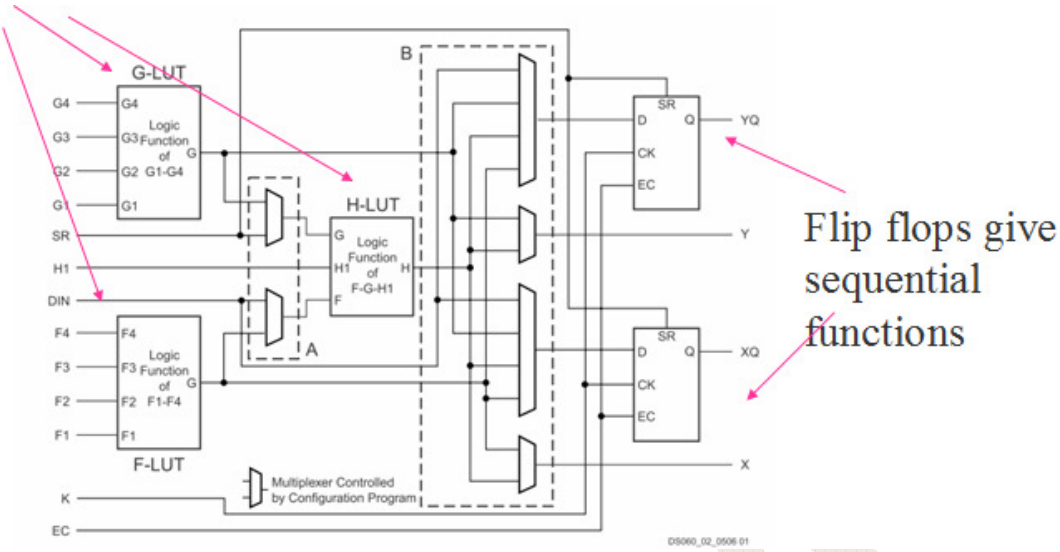
Change the CLBs to get the desired functions

5.1 Introduction

The Spartan-3E family architecture consists of five fundamental programmable functional elements:

Configurable Logic Blocks (CLBs) contain flexible Look-Up Tables (LUTs) that implement logic plus storage elements used as flip-flops or latches. CLBs perform a wide variety of logical functions as well as store data.

- **Input/output Blocks (IOBs)** control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Supports a variety of signal standards, including four high-performance differential standards. Double Data-Rate (DDR) registers are included.
- **Block RAM** provides data storage in the form of 18-Kbit dual-port blocks.
- **Multiplier Blocks** accept two 18-bit binary numbers as inputs and calculate the product.
- **Digital Clock Manager (DCM**) Blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase-shifting clock signals.



5.2 Programming a FPGA

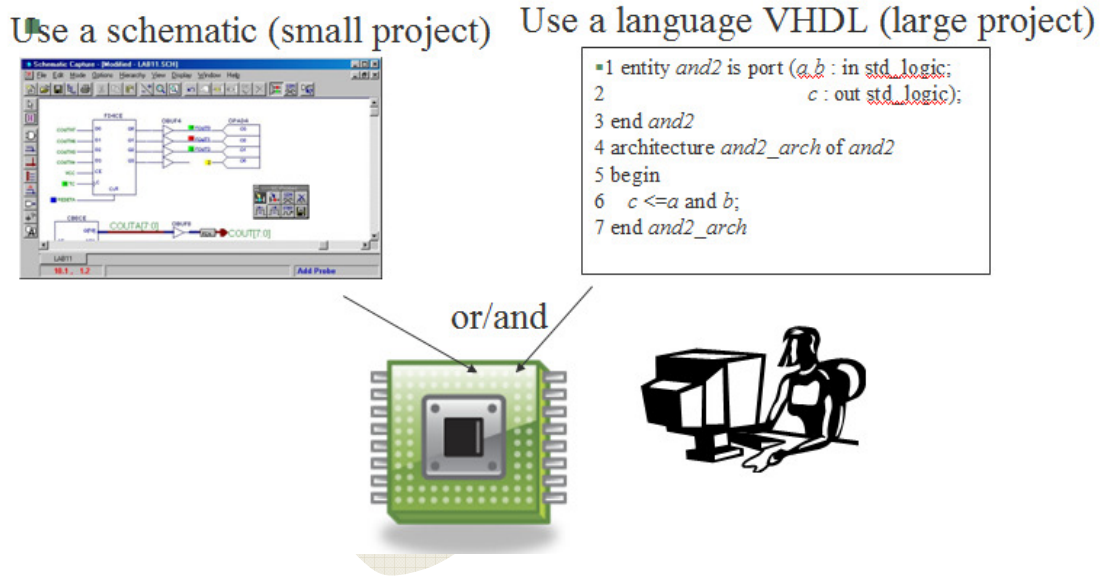
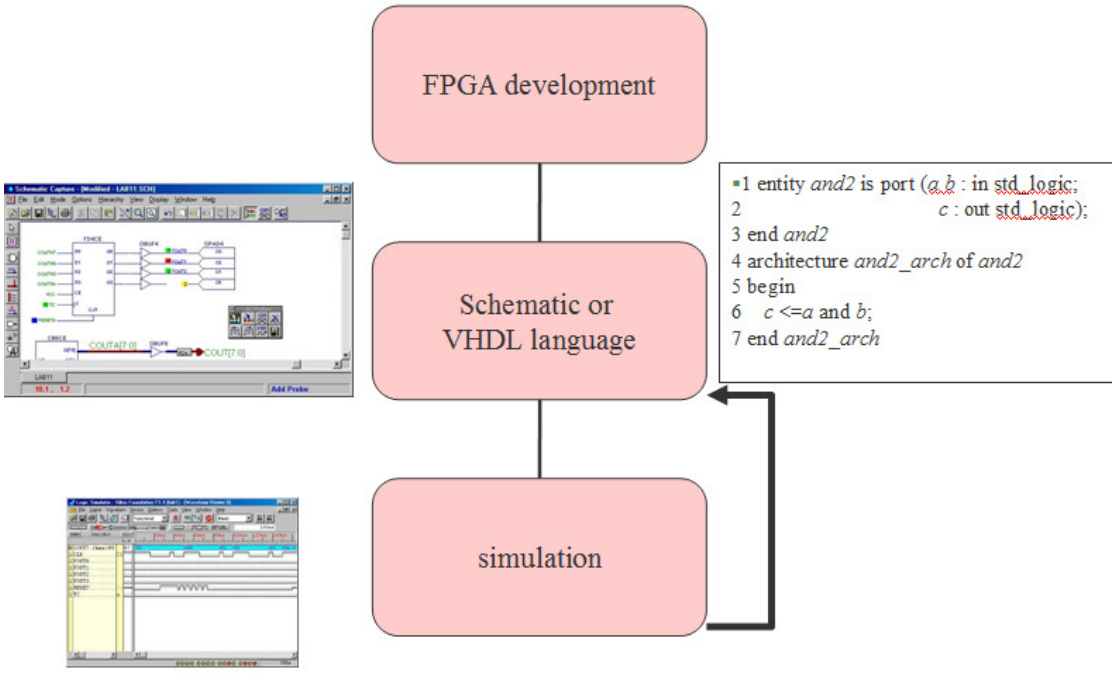
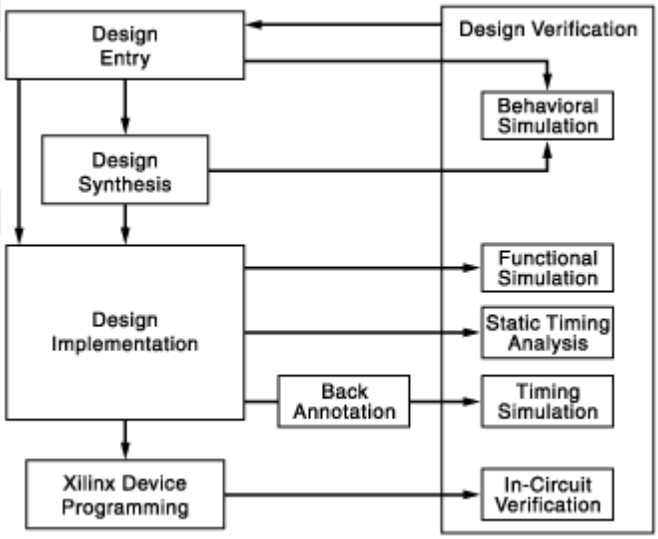


Figure 5.8



5.3 FPGA Design model



Design Entry

Create an ISE project as follows:

1. Create a project.
2. Create files and add them to your project, including a user constraints (UCF) file.
3. Add any existing files to your project.
4. Assign constraints such as timing constraints, pin assignments, and area constraints.

Functional Verification

You can verify the functionality of your design at different points in the design flow as follows:

- Before synthesis, run behavioral simulation (also known as RTL simulation).
- After Translate, run functional simulation (also known as gate-level simulation), using the SIMPRIM library.

- After device programming, run in-circuit verification.

Design Synthesis

Synthesize your design.

Design Implementation

Implement your design as follows:

1. Implement your design, which includes the following steps:

    - Translate
    - Map
    - Place and Route

2. Review reports generated by the Implement Design process, such as the Map Report or Place & Route Report, and change any of the following to improve your design:
    - Process properties
    - Constraints
    - Source files
3. Synthesize and implement your design again until design requirements are met.

Timing Verification

You can verify the timing of your design at different points in the design flow as follows:

- Run static timing analysis at the following points in the design flow:

- After Map
- After Place & Route
- Run timing simulation at the following points in the design flow:
- After Map (for a partial timing analysis of CLB and IOB delays)
- After Place and Route (for full timing analysis of block and net delays)

Xilinx Device Programming

Program your Xilinx device as follows:

1. Create a programming file (BIT) to program your FPGA.
2. Generate a PROM, ACE, or JTAG file for debugging or to download to your device.
3. Use iMPACT to program the device with a programming cable.

5.4 Tools Used In VLSI & EDA:

Electronic design automation (also known as EDA or ECAD) is a category of software tools for designing electronic systems such as printed circuit boards and integrated circuits. The tools work together in a design flow that chip designers use to design and analyze entire semiconductor chips. Before EDA, integrated circuits were designed by hand, and manually laid out...By the mid-70s, developers started to automate the design, and not just the drafting. The first placement and routing (Place and route) tools were developed. The proceedings of the Design Automation Conference cover much of this era. The next era began about the time of the publication of "Introduction to VLSI Systems" by Carver Mead and Lynn Conway in 1980. The earliest EDA tools were produced academically. One of the most famous was the "Berkeley VLSI Tools Tar ball", a set of UNIX utilities used to design early VLSI systems.

Xilinx ise8.2i

Xilinx, Inc. (NASDAQ: XLNX) is a supplier of programmable logic devices. It is known for inventing the field programmable gate array (FPGA) and as the first semiconductor company with a fabless manufacturing model.

Founded in Silicon Valley in 1984, , the company is headquartered in San Jose, California, U.S.A.; Dublin, Ireland; Singapore; and Tokyo, Japan. The company has corporate offices throughout North America, Asia and Europe.

Xilinx San Jose HQ Building at 2100 Logic Drive

Xilinx makes two main FPGA families: the high-performance Virtex series and the high-volume Spartan series, with a cheaper Easy Path option for ramping to volume production. The company also provides two CPLD lines, the Cool Runner and the 9500 series. In newer FPGA products, Xilinx minimized total power consumption by adopting a high-K metal gate (HKMG) process which allows for low static power consumption. At the 28nm node, static power is a significant portion of the total power dissipation of a device and in some cases is the dominant factor. Through the use of a HKMG process, Xilinx has reduced power use while increasing logic capacity.Virtex-6 and Spartan-6 FPGA families are said to consume 50 percent less power, and have up to twice the logic capacity compared to the previous generation of Xilinx FPGAs.

In June, 2010 Xilinx publically announced the introduction of the Xilinx 7 series, the Virtex-7, Kintex-7, and Artix-7 families, and promising improvements in system power, performance, capacity, and price. These new FPGA families are manufactured using TSMC's 28 nm HKMG process.

6. Appendix

6.1 Abbreviations Used

- FET - Field Effect Transistors
- HDL - Hardware Description Language
- VHDL - Very High Speed Integrated Circuit HDL
- IC - Integrated Circuits
- BJT - Bipolar Junction Transistors
- SSI - Small Scale Integration
- MSI - Medium Scale Integration
- LSI - Large Scale Integration
- VLSI - Very Large Scale Integration
- ULSI - Ultra Large Scale Integration
- MEMS - Micro Electro Mechanical Systems
- EDA - Electronic Design Automation
- ATPG - Automatic Test Pattern Generator
- BIST - Built In Self Test
- DSP - Digital Signal Processing
- MOS - Metal Oxide Semiconductor
- CMOS - Complimentary Metal Oxide Semiconductor
- DTL - Diode Transistor Logic
- ECL - Emitter Couple Logic
- TTL - Transistor Transistor Logic
- IIL - Integrated Injection Logic
- HTL - High Threshold Logic
- LAN - Local Area Network
- RTOS - Real Time Operating Systems
- PROM - Programmable Read Only Memory
- EPROM - Electronically Programmable Read Only Memory
- RAM - Random Access Memory
- ROM - Random Access Memory
- ADK - ASIC Design Kit
- ASIC - Application Specific Integrated Circuits
- LVS - Layout Versus Schematic
- DRC - Design Rule Check
- EDA - Electronic Design Automation

6.2 Program Codes:

```
----------------------------------------------------------------------------------
-- Company:      SRM University
-- Engineer:     Deepak Kumar Gupta
--               Anand Kumar Singh
--               Gautam Kr. Sinha
--               Anindita Panda
-- Create Date:    22:22:33 04/05/2011
-- Design Name: 32- Bit Microprocessor
-- Module Name: CPU
-- Project Name  32-Bit Microprocessor
-- Target Device: Spartan 3E
-- Tool versions: Xilinx 9.1
--               Modelsim SE 5.7f
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
```

--Cpu library

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;
package cpu_lib is
type t_shift is (shftpass,shl,shr,rotl,rotr);
subtype t_alu is unsigned(4 downto 0);
constant alupass : unsigned(4 downto 0) := "00000";
constant andop : unsigned (4 downto 0) := "00001";
constant orop : unsigned (4 downto 0) := "00010";
constant notop : unsigned(4 downto 0) := "00011";
constant xorop : unsigned (4 downto 0) := "00100";
constant plus : unsigned (4 downto 0) := "00101";
constant alusub : unsigned (4 downto 0) := "00110";
constant inc : unsigned (4 downto 0) := "00111";
constant dec : unsigned (4 downto 0) := "01000";
constant zero : unsigned (4 downto 0) := "01001";
type t_comp is (eq , neq, gt, gte, lt,lte );
subtype t_reg is std_logic_vector (4 downto 0);
type state is (reset1, reset2,reset3,reset4,reset5,reset6,execute,nop,load,store,move,
load2,load3,load4,store2,store3,store4,move2,move3,move4,incpc,incpc2,incpc3,incpc4,
incpc5,incpc6,loadpc,loadpc2,loadpc3,loadpc4,bgti2,bgti3,bgti4,bgti5,bgti6,bgti7,bgti8,
bgti9,bgti10,braii2,brai3,brai4,brai5,brai6,loadi2,loadi3,loadi4,loadi5,loadi6,inc2,inc3,
inc4);
subtype bit32 is std_logic_vector(31 downto 0);
```

```
end cpu_lib;
```

--**Alu**

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.cpu_lib.all;
entity alu is
   port (
      a: in bit32;
      b: in bit32;
      sel: in t_alu;
      c: out bit32;
                  reset: in bit
   );
end alu;
architecture behavioral of alu is
begin
 aluproc:process(a,b,sel)
 begin
 case sel is
 when alupass =>
 c<=a after 1 ns;
when andop=>
 c<=a and b after 1 ns;
when orop=>
 c<=a or b after 1 ns;
 when xorop=>
```

```
 c<=a xor b after 1 ns;
 when notop=>
 c<= not a after 1 ns;
 when plus=>
 c<= a + b after 1 ns;
 when alusub =>
 c<= a - b after 1 ns;
 when inc =>
 c <= a + "00000000000000000000000000000001";
 when dec =>
 c <= a - "00000000000000000000000000000001";
  when zero=>
 c<="00000000000000000000000000000000";
  when others=>
 c<="00000000000000000000000000000000";
 end case;
 end process; -- <<enter your statements here>>
end behavioral;
```

--**Comprator**

```
library IEEE ;
use IEEE.std_logic_1164.all;
use work.cpu_lib.all;
entity comp is
port(a,b:in bit32;
      sel:in t_comp;
      compout:out std_logic);
end comp;
```

```
architecture comp of comp  is
begin
process(a,b,sel)
begin
case sel is
when eq=>
 if(a=b)then
 compout<='1' after 1 ns;
 else
 compout<='0' after 1 ns;
 end if;
 when neq=>
 if(a/=b)then
 compout<='1' after 1 ns;
 else
 compout <='0' after 1 ns;
 end if;
when gt=>
 if(a>b)then
 compout <='1' after 1 ns;
 else
 compout<='0' after 1 ns;
 end if;
  when gte=>
 if(a>=b)then
 compout <='1' after 1 ns ;
 else
 compout <='0' after 1 ns;
```

```
end if;
when lt=>
if(a<b)then
compout<='1' after 1 ns;
else
compout<='0' after 1 ns;
end if;
when lte=>
if(a<=b)then
compout<='1' after 1 ns;
else
compout<='0' after 1 ns;
end if;
end case;
end process;
end comp;
```

--**Control unit**

```
library ieee;
use ieee.std_logic_1164.all;
use work.cpu_lib.all;
entity control is
port(reset,ready,clock,compout:in std_logic;
instrreg :in bit32;
progcntrwr,progcntrrd,addrregwr,outregwr,outregrd:out std_logic;
compsel: out t_comp;
alusel:out t_alu;
shiftsel:out t_shift;
```

```
regsel:out t_reg;
opregrd,opregwr,instrwr,regrd,regwr,rw,vma:out std_logic);
end control;
architecture control of control is
signal current_state , next_state :state;
begin
nxtstateproc:process(current_state,instrreg,compout,ready)
begin
progcntrwr <= '0';
progcntrrd <= '0';
addrregwr <= '0';
outregwr <= '0';
outregrd <= '0';
shiftsel <= shftpass;
alusel <= alupass;
compsel <= eq;
opregrd <= '0';
opregwr <='0';
instrwr <= '0';
regsel <= "00000";
regrd <= '0';
regwr <= '0';
rw <= '0';
vma <= '0';
case current_state is
when reset1=>
alusel<=zero after 1 ns;
shiftsel<=shftpass;
```

```
next_state<=reset2;
when reset2=>
alusel <= zero;
shiftsel <= shftpass;
outregwr <= '1';
next_state<=reset3;
when reset3=>
outregrd<='1';
next_state<=reset4;
when reset4=>
outregrd <='1';
progcntrwr<='1';
addrregwr<= '1';
next_state<=reset5;
when reset5 =>
vma <= '1';
rw <= '0';
next_state <= reset6;
when reset6 =>
vma <= '1';
rw <= '0';
if ready = '1' then
instrwr <= '1';
next_state <= execute;
else
next_state <= reset6;
end if;
when execute=>
```

```
case instrreg(31 downto 22) is
when "0000000000" =>    --nop
next_state <= incpc;
when "0000000001" => --load
regsel <= instrreg(10 downto 6);
regrd <= '1';
next_state <= load2;
when "0000000010" =>   --store
regsel <= instrreg(4 downto 0);
regrd <= '1';
next_state <=store2;
when "0000000011" =>   --move
regsel <= instrreg(10 downto 6);
regrd <= '1';
alusel <= alupass;
shiftsel <= shftpass;
next_state <= move2;
when "0000000100" =>          --loadi
progcntrrd <= '1';
alusel <= inc;
shiftsel <= shftpass;
next_state <=loadi2;
when "0000000101" =>             --branchimm
progcntrrd <= '1';
alusel <= inc;
shiftsel <= shftpass;
next_state <= braii2;
when "0000000110"  =>
```

```
regsel <= instrreg(10 downto 6);
regrd <= '1';
next_state <= bgti2;
when "0000000111" =>
regsel <= instrreg (4 downto 0);
regrd <= '1';
alusel<= inc;
shiftsel <=shftpass;
next_state <= inc2;
when others=>
next_state <= incpc;
end case;
when load2=>
regsel<=  instrreg(10 downto 6);
regrd<= '1';
addrregwr<= '1';
next_state<= load3;
when load3=>
vma <= '1';
rw <= '0';
next_state<= load4;
when load4=>
vma <= '1';
rw <= '0';
regsel <= instrreg(4 downto 0);
regwr <= '1';
next_state <=incpc;
when store2=>
```

```
regsel <= instrreg (4 downto 0);
regrd <= '1';
addrregwr <= '1';
next_state <= store3;
when store3=>
regsel <= instrreg(10 downto 6);
regrd <= '1';
next_state <= store4;
when store4 =>
regsel <= instrreg(10 downto 6);
regrd<='1';
vma <='1';
rw <= '1';
next_state <= incpc;
when move2 =>
regsel<= instrreg(10 downto 6);
regrd<= '1';
alusel<=alupass;
shiftsel<=shftpass;
outregwr <= '1';
next_state <= move3;
when move3 =>
outregrd <= '1';
next_state <= move4;
when move4=>
outregrd <= '1';
regsel <= instrreg(4 downto 0);
regwr <= '1';
```

```
next_state <= incpc;
when loadi2 =>
progcntrrd <= '1';
alusel <= inc;
shiftsel <= shftpass;
outregwr <= '1';
next_state <= loadi3;
when loadi3 =>
outregrd<= '1';
next_state <= loadi4;
when loadi4 =>
outregrd <= '1';
progcntrwr <='1';
addrregwr <= '1';
next_state<= loadi5;
when loadi5 =>
vma <= '1';
rw <= '0';
next_state <= loadi6;
when loadi6=>
vma <= '1';
rw <= '0';
  if(ready='1') then
regsel<= instrreg(4 downto 0);
regwr<= '1';
next_state<= incpc;
else
next_state<= loadi6;
```

```
end if ;
when braii2 =>
progcntrrd <= '1';
alusel<= inc;
shiftsel<= shftpass;
outregwr<= '1';
next_state<= brai3;
when brai3=>
outregrd<= '1';
next_state<= brai4;
when brai4=>
outregrd<= '1';
progcntrwr <= '1';
addrregwr <= '1';
next_state<= brai5;
when brai5=>
vma <= '1';
rw <= '0';
next_state <= brai6;
when brai6 =>
vma <= '1';
rw <= '0';
if(ready='1')then
progcntrwr<='1';
next_state<= loadpc;
else
next_state<= brai6;
end if;
```

```
when bgti2=>
regsel<= instrreg(10 downto 6);
regrd<= '1';
opregwr <= '1';
next_state<= bgti3;
when bgti3 =>
opregrd <= '1';
regsel <= instrreg (4 downto 0);
regrd <= '1' ;
compsel <= gt;
next_state <= bgti4;
when bgti4 =>
opregrd <= '1' after 1 ns;
regsel <= instrreg (4 downto 0);
regrd <= '1';
compsel <= gt;
if compout = '1' then
next_state <= bgti5;
else
next_state <= incpc;
end if;
when bgti5 =>
progcntrrd <= '1';
alusel <= inc;
shiftsel <= shftpass;
next_state <= bgti6;
when bgti6 =>
progcntrrd <= '1';
```

```
alusel <= inc;
shiftsel <= shftpass;
outregwr <= '1';
next_state <= bgti7;
when bgti7 =>
outregrd <= '1';
next_state <= bgti8;
when bgti8 =>
outregrd <= '1';
progcntrwr <= '1';
addrregwr <= '1' ;
next_state <= bgti9;
when bgti9=>
vma <= '1';
rw <= '0';
next_state <= bgti10;
when bgti10 =>
vma <= '1';
rw <= '0';
if ready = '1' then
progcntrwr <= '1';
next_state <= loadpc;
else
next_state <= bgti10;
end if;
when inc2 =>
regsel <= instrreg (4 downto 0);
regrd <= '1';
```

```
alusel <= inc ;
shiftsel<= shftpass;
outregwr <= '1';
next_state <= inc3;
when inc3 =>
outregrd <='1';
next_state <= inc4;
when inc4 =>
outregrd <= '1';
regsel <= instrreg(4 downto 0);
regwr <= '1';
next_state <= incpc;
when loadpc =>
progcntrrd <= '1' ;
next_state <= loadpc2;
when loadpc2 =>
progcntrrd <= '1' ;
addrregwr <= '1';
next_state <= loadpc3;
when loadpc3 =>
vma <= '1';
rw <= '0';
next_state <= loadpc4;
when loadpc4 =>
 vma <= '1';
rw <= '0';
if (ready ='1') then
instrwr <= '1';
```

```
next_state <= execute ;
else
next_state <= loadpc4;
end if;
when incpc =>
progcntrrd <= '1';
alusel <= inc;
shiftsel <= shftpass;
next_state <= incpc2;
when incpc2 =>
progcntrrd <= '1';
alusel <= inc;
shiftsel <= shftpass;
outregwr <= '1';
next_state <= incpc3;
when incpc3 =>
outregrd <= '1';
next_state <= incpc4;
when incpc4 =>
outregrd <= '1';
progcntrwr <= '1';
addrregwr <= '1';
next_state <= incpc5;
when incpc5 =>
vma <= '1';
rw <= '0';
if (ready = '1') then
instrwr <= '1';
```

```
next_state <= execute;
else
next_state <= incpc6;
end if;
when others =>
next_state <=incpc;
end case;
end process;
controlffroc: process(clock,reset)
begin
if (reset ='1') then
current_state <= reset1 after 1 ns;
elsif clock'event and clock = '1' then
current_state <= next_state  after 1 ns;
end if;
end process;
end control;
```

--**Register**

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.cpu_lib.all;
entity reg is
   port (
      a: in bit32;
      clk: in std_logic;
      q: out bit32    );
end reg;
```

```
architecture behavioral of reg is
begin
regproc:process
begin
wait until clk'event and clk = '1';
q<= a after 1 ns ;
end process;
end behavioral;
```

--**Register array**

```
library IEEE;
use IEEE.std_logic_1164.all;
use  ieee.std_logic_unsigned.all;
use work.cpu_lib.all;
entity regarray is
   port (data : in bit32;
sel: in t_reg;
en: in std_logic;
clk: in std_logic;
q:out bit32);
end regarray;
architecture behavioral of regarray is
type t_ram is array (0 to 15) of bit32;
signal temp_data: bit32 ;
begin
process (clk,sel)
variable ramdata : t_ram;
begin
```

```
if clk'event and clk = '1' then
ramdata(conv_integer(sel)):=data;
end if;
temp_data <= ramdata(conv_integer(sel)) after 1 ns;
end process;
process(en,temp_data)
begin
if en = '1' then
q<= temp_data after 1 ns;
else
q <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ" after 1 ns;
end if;
end process;
end behavioral;
```

--**Shift Unit**

```
library IEEE;
use work.cpu_lib.all;
use IEEE.std_logic_1164.all;
entity shift is
port(a:in bit32;
     sel:in t_shift;
     y:out bit32);
end shift;
architecture behavioral of shift is
begin
shiftproc:process(a,sel)
begin
```

```
case sel is
when shftpass=>
y<=a after 1 ns;
when shl=>
y<= a(30 downto 0) & '0' after 1 ns;
when shr=>
y<= '0' & a(31 downto 1) after 1 ns;
when rotl=>
y<= a(30 downto 0) & a(31) after 1 ns;
when rotr=>
y<= a(0) & a(31 downto 1) after 1 ns;
end case;
end process;
end behavioral;
```

--**Triregister**

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.cpu_lib.all;
entity trireg is
   port (
      a: in bit32;
      en: in STD_LOGIC;
     clk: in STD_LOGIC;
      q: out bit32    );
end trireg;
architecture behavioral of trireg is
signal val: bit32;
```

```
begin
 triregdata:process
 begin
 wait until clk'event and clk = '1' ;
 val<=a;
 end process;
 trireg3st:process(en,val)
 begin
 if en= '1' then
 q<= val after 1 ns;
 elsif en = '0' then
 q <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ" after 1 ns;
 else
 q <= "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" after 1 ns;
 end if;
 end process;
 end behavioral;
```

--**Cpu**

```
entity cpu is
   port (
      vma: out STD_LOGIC;
      rw: out STD_LOGIC;
      clock: in STD_LOGIC;
      reset: in STD_LOGIC;
      ready: in STD_LOGIC;
      addr: out bit32;
      data: inout bit32);
```

```
end cpu;
architecture behavioral of cpu is
component reg
port(a: in bit32;
clk: in std_logic;
q:out bit32);
end component;
component trireg
port( a: in bit32;
en,clk:in std_logic;
q: out bit32);
end component;
component control is
port(clock:in std_logic;
reset:in std_logic;
instrreg:in bit32;
compout:in std_logic;
ready:in std_logic;
progcntrwr: out std_logic;
progcntrrd: out std_logic;
addrregwr: out std_logic;
outregwr: out std_logic;
outregrd: out std_logic;
shiftsel:out t_shift;
alusel:out t_alu;
compsel:out t_comp;
opregrd: out std_logic;
opregwr:out std_logic;
```

```
instrwr:out std_logic;
regsel:out t_reg;
regrd:out std_logic;
regwr:out std_logic;
rw:out std_logic;
vma:out std_logic);
end component;
component alu
port( a,b:in bit32;
sel:in t_alu;
c:out bit32);
end component;
component shift is
port(a: in bit32;
sel :in t_shift;
y:out bit32);
end component;
component comp
port(a,b: in bit32;
sel:in t_comp;
compout:out std_logic);
end component;
component regarray
port(data : in bit32;
sel: in t_reg;
en: in std_logic;
clk: in std_logic;
q:out bit32);
```

```
end component;

signal
regrd,regwr,opregrd,opregwr,outregrd,outregwr,addrregwr,instrregwr,progcntrrd,progcntrwr,compout:
std_logic ;

signal opdata,aluout,shiftout,instrregout:bit32;

signal shiftsel:t_shift;

signal alusel:t_alu;

signal regsel :t_reg;

signal compsel:t_comp;

begin

ra1: regarray

port map(data,regsel,regrd,regwr,data);

opreg: trireg  --  pc

port map(data,opregrd,opregwr,opdata);

u4:trireg   -- outreg

port map(shiftout,outregrd,outregwr,data);

alu1: alu

port map(data,opdata,alusel,aluout);

shift1:shift

port map(aluout,shiftsel,shiftout);

outreg:trireg   -- outreg

port map(shiftout,outregrd,outregwr,data);

addrreg:reg  -- addreg

port map(data,addrregwr,addr);

progcntr: trireg

port map (data, progcntrrd,progcntrwr,data);

comp1:comp

port map(opdata,data,compsel,compout);
```

```
instr:reg  --  instr reg

port map(data,instrregwr,instrregout);

con1:control

port map(clock,reset,instrregout,compout,ready,

progcntrwr,progcntrrd,addrregwr,outregwr,outregrd,shiftsel,alusel,compsel,opregrd,opregwr,instrregwr,regsel,regrd,regwr,rw,vma);

end behavioral;
```

--Memory

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.cpu_lib.all;

entity mem is

port(addr:in bit32;

sel,rw:in std_logic;

ready:out std_logic;

data:inout bit32);

end mem;

architecture Behavioral of mem is

begin

memproc:process(addr,sel,rw)

type t_mem is array(0 to 63) of bit32;

variable mem_data:t_mem:=("00000000000000000010000000000001",--- 0 loadi 1,# --load source

"00000000000000000000000000010000",---1  10

"00000000000000000010000000000010",---2 loadi 2, # --load

"00000000000000000000000000110000",---3 30

"00000000000000000010000000000110" ,---4 loadi 6, # --load end
```

"00000000000000000000000000101111",---5 2F

"00000000000000000000100000001011",---6 load 1,3--load reg3

"00000000000000000001000000011010",---7 store 3,2 --store reg3

"00000000000000000011000000001110",---8 bgti 1,6,#---compare to

"00000000000000000000000000000000",---9 00 --if so just

"00000000000000000111000000000001",---A inc 1 --move source

"00000000000000000111000000000010",--B inc 2 --move

"00000000000000000101000000001111",---C brai # --go to the

"00000000000000000000000000000110",--D 06

"00000000000000000000000000000000",--E

"00000000000000000000000000000000",--F

"00000000000000000000000000000001",--10 --- start of source

"00000000000000000000000000000010",--11

"00000000000000000000000000000011",--12

"00000000000000000000000000000100",--13

"00000000000000000000000000000101",--14

"00000000000000000000000000000110",--15

"00000000000000000000000000000111",--16

"00000000000000000000000000001000",--17

"00000000000000000000000000001001",--18

"00000000000000000000000000001010",--19

"00000000000000000000000000001011",--1A

"00000000000000000000000000001100",--1B

"00000000000000000000000000001101",--1C

"00000000000000000000000000001110",--1D

"00000000000000000000000000001111",--1E

"00000000000000000000000000010000",--1F

"00000000000000000000000000000000",--20

"00000000000000000000000000000000",--21

"00000000000000000000000000000000",--22

"00000000000000000000000000000000",--23

"00000000000000000000000000000000",--24

"00000000000000000000000000000000",--25

"00000000000000000000000000000000",--26

"00000000000000000000000000000000",--27

"00000000000000000000000000000000",--28

"00000000000000000000000000000000",--29

"00000000000000000000000000000000",--2A

"00000000000000000000000000000000",--2B

"00000000000000000000000000000000",--2C

"00000000000000000000000000000000",--2D

"00000000000000000000000000000000",--2E

"00000000000000000000000000000000",--2F

"00000000000000000000000000000000",--30 ---start of destination

"00000000000000000000000000000000",--31

"00000000000000000000000000000000",--32

"00000000000000000000000000000000",--33

"00000000000000000000000000000000",--34

"00000000000000000000000000000000",--35

"00000000000000000000000000000000",--36

"00000000000000000000000000000000",--37

"00000000000000000000000000000000",--38

"00000000000000000000000000000000",--39

"00000000000000000000000000000000",--3A

"00000000000000000000000000000000",--3B

"00000000000000000000000000000000",--3C

```
"00000000000000000000000000000000",--3D
"00000000000000000000000000000000",--3E
"00000000000000000000000000000000");--3F
begin
data <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
ready <= '0';
if sel = '1' then
if rw = '0' then
data <= mem_data(conv_integer(addr(31 downto 0)))    after 1 ns;
ready<= '1';
elsif rw = '1' then
mem_data(conv_integer(addr(15 downto 0))) := data;
end if;
else
data <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ" after 1 ns;
end if;
end process;
end Behavioral;
```

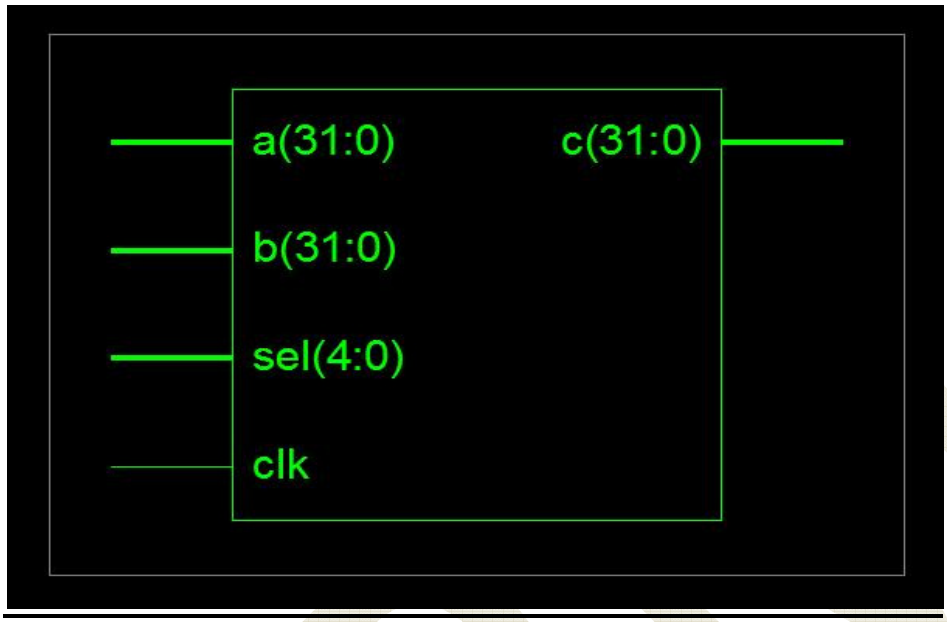
--**Top level system design**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.cpu_lib.all;
entity top is
end top;
architecture Behavioral of top is
```

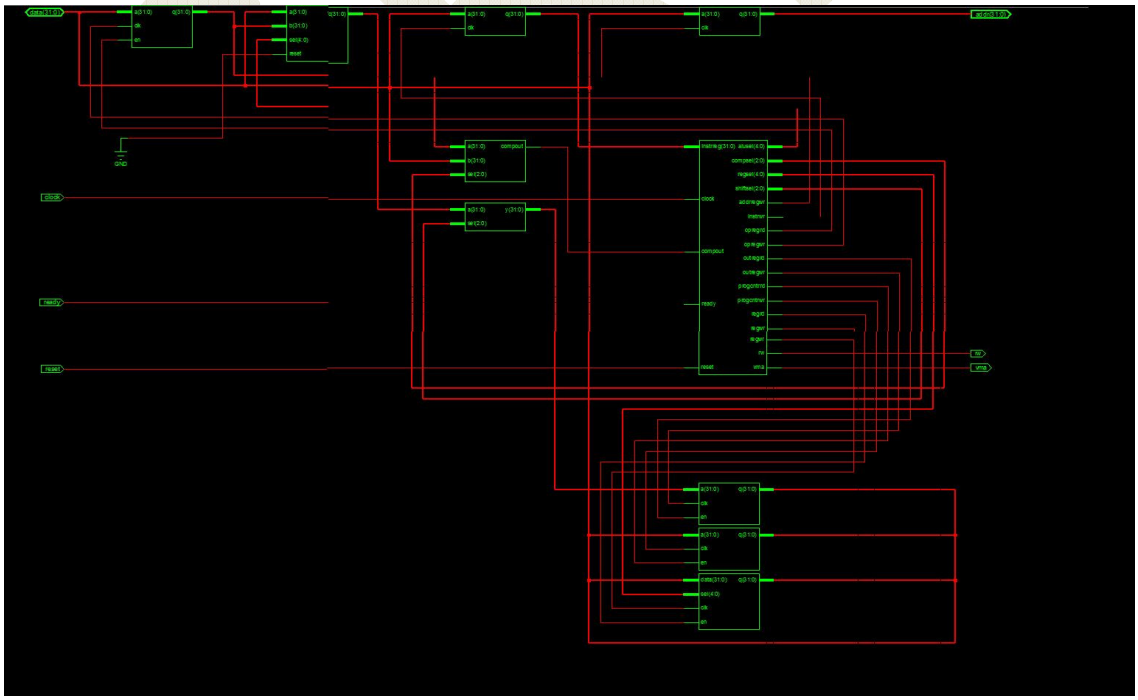
```
component mem
port (addr:in bit32;
sel,rw:in std_logic;
ready:out std_logic;
data :inout bit32);
end component;
component cpu
port (clock,reset,ready:in std_logic;
addr:out bit32;
rw,vma:out std_logic;
data:inout bit32);
end component;
signal addr, data: bit32;
signal vma,rw,ready:std_logic;
signal clock,reset:std_logic:='0';
begin
clock <= not clock after 50 ns;
reset <= '1' , '0' after 100 ns;
m1:mem port map (addr,vma,rw,ready,data);
u1:cpu port map(clock,reset,ready,addr,rw,vma,data);
end behavioral;
```

7. Results

7.1 Schematic



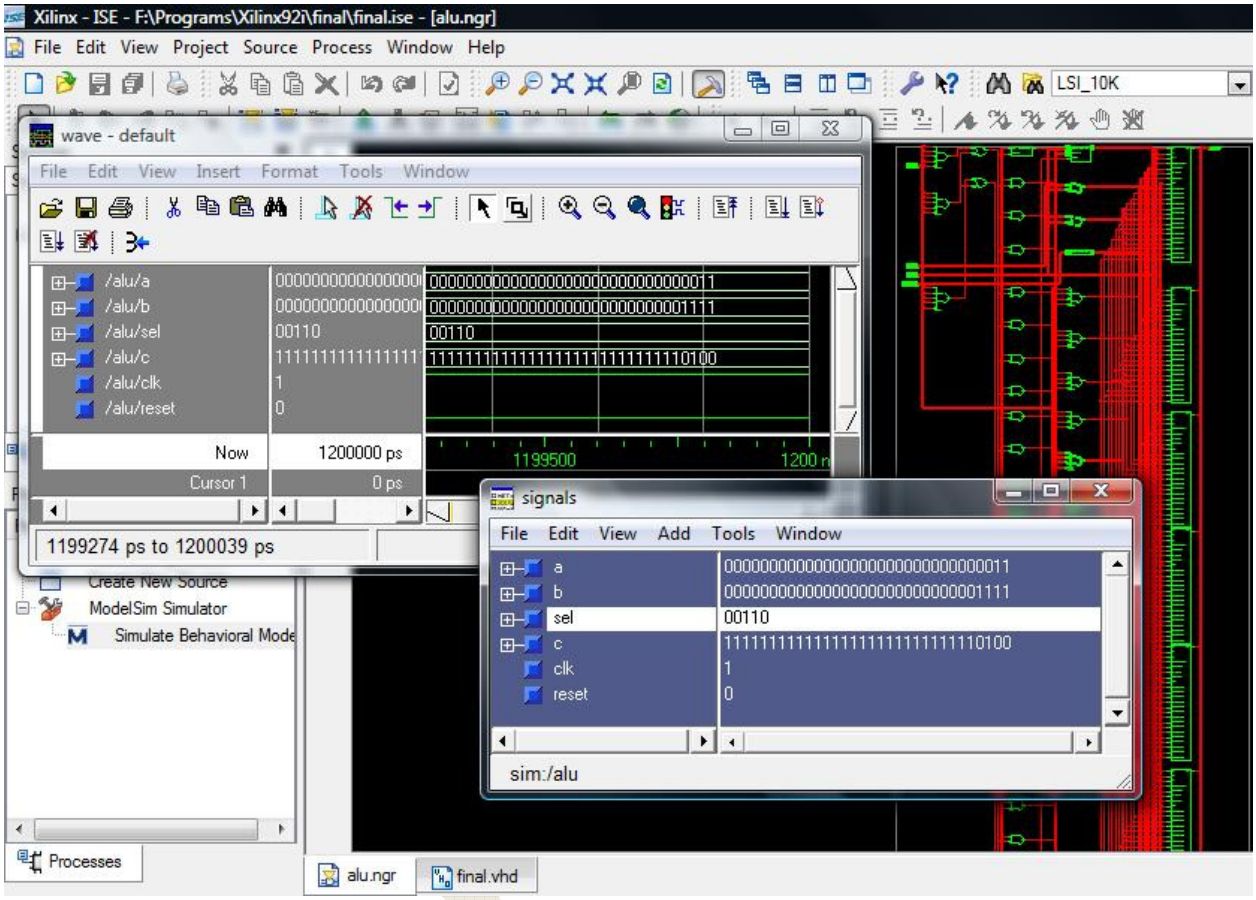
7.2 RTL View:



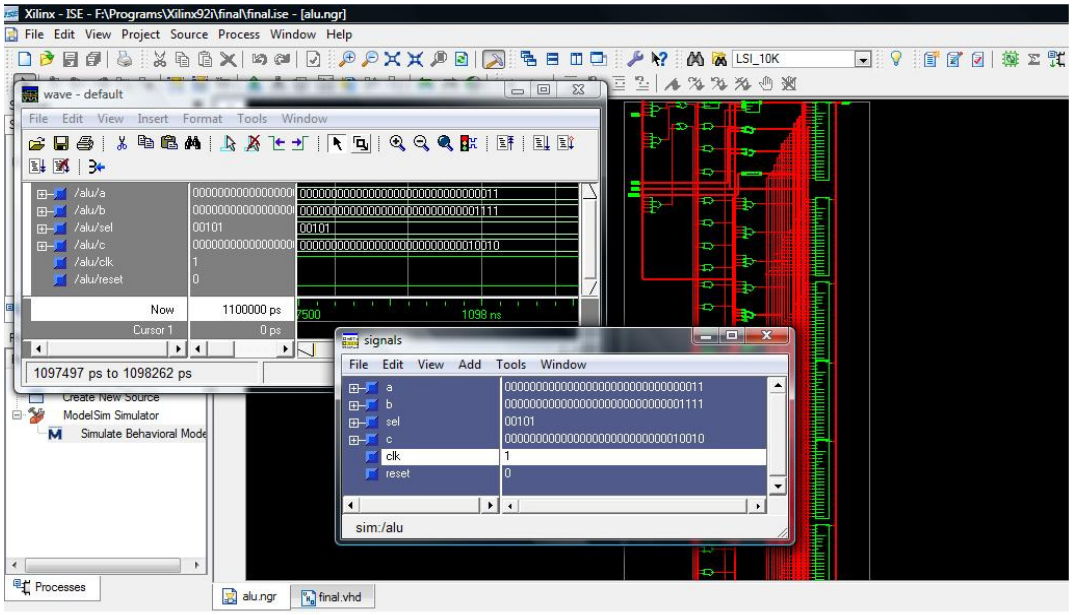
8. Conclusion:

Thus the output waveform has been verified in accordance with the inputs given.

Output1:



Output2


Xilinx - ISE - F:\Programs\Xilinx92i\final\final.ise - [alu.ngr]
File Edit View Project Source Process Window Help
wave - default
File Edit View Insert Format Tools Window
/alu/a
/alu/b
/alu/sel
/alu/c
/alu/clk
/alu/reset
00101
Now
1100000 ps
Cursor 1
0 ps
1097497 ps to 1098262 ps
signals
File Edit View Add Tools Window
a
b
sel
c
clk
reset
sim:/alu
ModelSim Simulator
Simulate Behavioral Mode
Processes
alu.ngr
final.vhd

8.2 Uses of microprocessors

- Most appliances found around the house are operated by microprocessors.
- Most modern factories are fully automated—that means that most jobs are done by a computer.
- Automobiles, trains, subways, planes, and even taxi services require the use of many microprocessors.
- Another use is in the suspension system. A processor, controls the amount of pressure applied to keep the car leveled. During turns, a processor, slows down the wheels on the inner side of the curb and speeds them up on the outside to keep the speed constant and make a smooth turn.
- In space explorations.
- In solving various complex scientific computations.

9. References:

- D. Gajski and R. Khun , "Introduction: New VLSI Tools," IEEE Computer, Vol. 16, No. 12, pp. 11-14, Dec.1983.
- J. Bhasker, "VHDL Primer", VHDL Primer",3rd Edition, Prentice Hall , Upper Saddle River, 1998.
- P. J. Ashenden, "The Student's Guide to VHDL", Morgan Kaufmann Publishers, Inc, San Francisco,1998.
- C. H. Roth, "Digital System Design using VHDL", PWS Publishing Company, New York,1998.
- D. Pellerin and D. Taylor, " VHDL Made Easy!", Prentice Hall , Upper Saddle River,1997.
- Volnei A. Pedroni, "Circuit Design with VHDL", PHI, 2005.
- Xilinx website:- http://www.xilinx.com/
  http://toolbox.xilinx.com/docsan/data/fndtn/vhd/vhd.htm
- B.Ram "Fundamentals of Microprocessor and Microcomputer", Dhanpat Rai Publication.
- P.Raja "Microprocessor and Applications", Umesh Publication.
- Wikipedia.com
- Google.com