“RTL DESIGN OF RISC PROCESSOR FOR DSP APPLICATION”

Major Project Report

Submitted in Partial Fulfillment of the Requirements for

Degree
of

Master of Technology

In
Electronics & Communication Engineering
(VLSI Design)

By
Jayesh J. Patel
(05MEC011)

Under the Guidance of
Dr K. S. Dasgupta
Group Director (ADCTG),
SAC (ISRO)-Ahmedabad

Electronics & Communication Engineering Department Institute of Technology
Nirma University of Science & Technology
Ahmedabad 382 481
May 2007

CERTIFICATE

This is to certify that the Major Project Report entitled **“RTL Design of RISC Processor for DSP Application" submitted** by **Jayesh J Patel** (Roll No. **05MEC011)** as the partial fulfilment of the requirements for award of the degree of M.Tech (EC-VLSI Design) awarded by institute of Technology, Nirma **University,** Ahmedabad embodies work carried out by him under my supervision at **Space Application Centre (ISRO),** Ahmedabad, Gujarat during w.e. f SEPTEMBER 2006-APRIL 2007.

Date:
Place:

Dr. K. S. Dasgupta
Group Director-ADCTG
SAC (ISRO)
Ahmedabad

CERTIFICATE

This is to certify that the Major Project Report entitled **"RTL Design of RISC Processor for DSP Application"** submitted by **Jayesh J Patel (**Roll No. **05MEC011)** as the partial fulfilment of the requirements for the award of the degree of Master of Technology in Electronics & Communication **(VLSI Design)** of Institute of Technology **Nirma University** is the record of work carried out by him under my/our supervision and guidance. The work submitted has in our opinion reached a level required for being accepted for the examination. The results embodied in this major project work to the best of our knowledge have not been submitted to any other University or Institution for award of any degree or diploma.

Date :
Place:

Facilitator at Institute

Prof. N. P. Gajjar
Nirma University
Ahmedabad

Dr. N. M. Devashrayee
PG Coordinator-VLSI Design

Prof. A. S. Ranade
HOD
EC-Department
Nirma University
Ahmedabad

Prof. A. B. Patel
Director
Institute of Technology,
Nirma University
Ahmedabad

ACKNOWLEDGEMENT

It has been great pleasure for me in doing a major project work on RTL DESIGN OF RISC PROCESSOR FOR DSP APPLICATION under Guidance of Dr. K. S. Dasgupta Group Director, SAC. I am very grateful to him who assigned me a project under his expert guidance in the ACTD/ADCTG Department, without his invaluable guidance the work would have not been possible. His academic excellence continues to be a source of inspiration. He always inspires us to put best efforts to achieve the goal.

I would like to express my sincere thanks to Prof. N. P. Gajjar for his kind, ordinal and valuable guidance and support at every moment of the project His practical approach, needful help encouraged me to do better work. His dedication to work always inspired me to do task as a challenge.

I like to express thanks to Director, Dr. N. M. Devashrayee, PG coordinator-VLSI Design and Institute for providing me such nice opportunity to do my dissertation work at the prime organisation like SAC (ISRO) Ahmedabad.

I express my gratitude to Mr. RJK Jain Head, HRD department (SAC) who accepted my candidature for dissertation work at SAC, ISRO Ahmedabad. I am very much thankful to Shri Pinakin Thaker who has rendered their guidance and extended co-operation at all times.

Thanks to my family members for their faith, colleagues for giving me support. Finally thanks to God for helping me out at difficult times.

Jayesh J. Patel
05MEC011

ABSTRACT

In order to achieve better performance of processor, we should strive to achieve a situation where most of the featured instructions are executable in a single cycle. Ideally, we would like to see a streamlined and uniform handling of all instructions, where the fetch and the execute stages take up the same time for any instruction, desirably, a single clock. This is basically one of the first and most important principles inherent in the RISC design. The RISC has attributes like Simple instructions, less complexity, Compiler generates software routines to perform complex instructions, and Instruction size is constant

The project aims at providing RTL Design of RISC processor for DSP application, which may be implemented on FPGA.

Various processor cores like ARM processor NIOS_II processor, PowerPC etc. are available in market which are widely acceptable by industries. These processors are available in form of Soft IP core module

The project work includes detailed literature survey of ARM processor, NIOS_II processor, PowerPC. Comparison of these processors is carried out based on various parameters like Performance, Code density, Strength of instruction set, Supports by software tools, Interrupt mechanism etc.

In this work, 32-bit processor is designed using VHDL. The designed processor supports 32 various instructions. It supports all general addressing modes. The functionality of the designed is verified by writing a program to transfer a block of data in memory.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated circuit
CISC	Complex Instruction Set Computer
CPLD	complex programmable logic Device
CPSR	Current Program Status Register
DMA	Direct Memory Access
DSP	Digital signal Processing
FPGA	Field Programmable Gate Array logic
IDE	Integrated Development Environment
IRL	Internet Reconfigurable Logic
LCD	liquid Crystalline Display
MIPS	million Instructions per second
MMU	Memory Management Unit
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SOC	System-On-Chip
SOPC	System-On-a-Programmable-Chip
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VLSI	Very Large Scale Integration
UART	Universal Asynchronous receiver transmitter
VMA	Valid Memory Address

ABOUT ISRO

ABOUT ORGANIZATION:

The Indian Space Program was formally organized in 1972 when Government of India set up the development and application of space technology and space sciences for the socioeconomic benefit of the commission include the formulation if the policy of the Department of Space and implementation of the Government's policy in all matters concerning outer space.

The Indian Space Research Organization (ISRO) under DOS plays a key role, through its centers, in the planning and execution of National Space activities. It is also responsible for technical management in the area of Space Application and space technology. Basic natural resources survey, and meteorology, other R&D activities of ISRO in satellite achieving the basic application objectives

OBJECTIVES OF ISRO

- Long distance telecommunication, diffusion of TV signals using satellite. Remotes sensing of natural and renewable earth resources and meteorological parameters from satellites.
- Satellite based resources survey, management and environment monitoring.
- Development and operationalisation of indigenous satellite.
- To realize the above objectives of ISRO activities are oriented predominantly towards design and development of applications satellites for communications, Remote Sensing, TV broadcasting and meteorology.
- Design and Development of Rocket launching vehicles to place these application satellites into the required orbits.
- Establishment of ground stations/facilities for using these satellites and for launching.

The headquarters of both DOS and ISRO located at Bangalore provides overall direction to the technical, scientific and administrative functions of the ISRO centers/units. These are as follows

Satellite Launching Technology Development Centers:

- Vikram Sarabhai Space Centre (VSSC)-Trivandrum.
- SHAR Centre-Sriharikota.
- ISRO Telemetry, Tracking and Command Network (ISTRAC)-Bangalore.
- ISRO Satellite Centre (ISAC)-Bangalore.

SAC, Ahmedabad is ISRO's application R& D centre. The primary responsibilities of the centre are to conceptualize plan and execute projects and research programmes leading to practical applications of space technology. The main area of activities are satellite based telecommunication, and Remote Sensing for natural resources survey and management, environment test facilities and reliability and quality assurance. SAC also manages the DELHI earth station.

The major tasks of this centre are to conceptualize and conduct research and development, and execute projects in the field of space application. To this end, SAC has two board areas of activity viz., satellite based communication, including television, and remote sensing for natural resources survey and management, meteorology and geodesy.

To carry out its programmes, the centre is organized functionally in to the communication area and the remote sensing area. The technical service group, test, evaluation, standards and calibration facility and Payload Fabrication Facility provide the support services.

Some of the major facilities at SAC include:

Experimental Satellite Communication Earth Station at Ahmedabad and Delhi.

Transportable Remote Area Communication Terminal (TRACT).

Emergency Communication Terminal (ETC).

Small Communication Terminal (SCOT).

Meteorological Data Reception Terminal.

Electronics and Mechanical Fabrication.

Environmental and Space Simulation.

Visual Photo Interpretation.

Digital Image processing computer.

Antenna Test Range.

INTRODUCTION TO ADCTG

It is responsible for the technology development related to on board signal processing required for future SATCOM services. ADCTG consists of the following two divisions.

On Board Signal Processing Division (OSPD)

OSPD is responsible for the development of on board signal processing technology for future SATCOM services.

Advanced Communication Technology Division (ACTD)

ACTD carry out advanced R&D for satellite based digital communication & secured communication. It is also responsible for technology development related to wide band networks.

ABOUT FACILITY

As ISRO is an R & D organization, it is very healthy in form of laboratory and in form of Equipment's facility, we have separate Computer machine with required software. (e. g. Matlab, Xilinx. etc.). There is good facility of library, which contains almost all-technical literature in form of books, journals, research papers, and transactions.

CHAPTER 1
INTRODUCTION

1.1 Motivation for Thesis

In its relatively brief lifetime of more than thirty years, the microprocessor has undergone phenomenal advances. Its performance has improved at the outstanding rate of doubling every 18months. In past three decades, microprocessors have been responsible for inspiring and facilitating of the major innovations in the computer systems. Design, which can be implemented on FPGA, offers the great advantage of flexibility. Soft processor is the processor that can be implemented on FPGA and value addition is possible in future.

1.1.1 Aim

My project titled as **"RTL Design of RISC processor for DSP Application".** RISC processor expects to perform more of the optimizations via the compiler. The purpose of doing is this is to reduce the hardware complexity and thus achieve a much faster operation.

Milestones

- Literature review and understanding processor functionality and its architecture
- Study of various core processors like ARM, NIOS_II, PowerPC etc.
- RTL Design of 32-bit processor
- Prototype system design based on processor and memory
- Implementation of system on Xilinx based FPGA

1.2 Background

Processor is key component for any embedded system. Embedded system is combination of hardware and software. The project related to the embedded system is based on the knowledge of Hardware/ software co-design. The project carried out is belongs to embedded system field. It is embedded system design for specific DSP application.

There are various processor cores like ARM, NIOS, POWERPC, MICROBLAZE, PICOBLAZE are widely used for embedded system. The study of some of such processors has been completed during the first phase of project work. Mainly detailed study of features of ARM and NIOS_II processor is carried out. The comparative study is presented in the report.

System level design work is carried out during major project. The design may be implemented on FPGA. FPGA is programmable logic device, which is a general-purpose chip for implementing logic circuitry. It contains a collection of logic circuit elements that can be tailored or customized in different ways. A PLD can be viewed as a "Black Box" that contains logic gates and programmable switches, as shown in Figure-1.1.

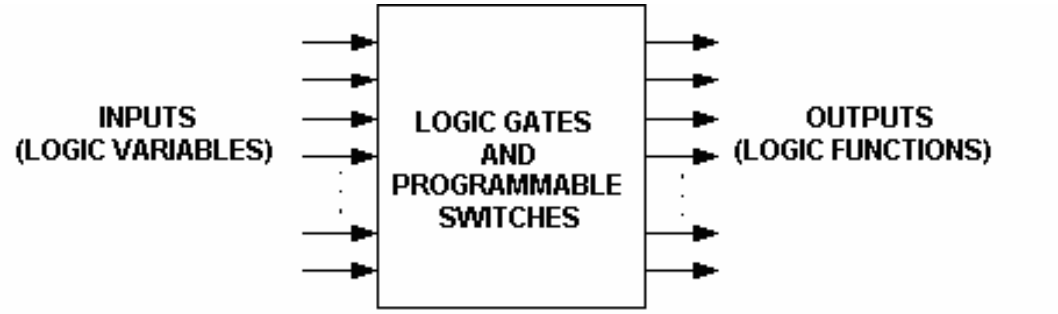

Figure-1.1 programmable logic device as black box

The programmable switches allow the logic gates inside the PLD to be connected together to implement whatever logic circuit is needed. This makes the real estate requirement on the printed circuit board required is very small. Therefore the problems associated with longer signal tracks ground returns are eliminated.

Field Programmable Gate Arrays (FPGAs) provide a rapid prototyping platform. FPGAs are devices that can be reprogrammed to achieve different functionality without incurring the non-recurring engineering costs typically associated with custom IC fabrication. In this work, the target platform is the Xilinx Virtex FPGA

RTL coding is done using VHDL as hardware description language. Structural modeling concept is used. ALU, shift register, register array, tristate register, and instruction decoder are designed. Interfacing is carried out on single bus (32-bit)

1.3 Organization of Thesis

In this Thesis Chapter 2 gives a background discussion on RISC processor. Chapter 3 introduces ARM processors. The study of NIOS II processor and PowerPC processor is represented in Chapter 4 and Chapter 5 respectively. Chapter 6 gives a comparison of ARM and NIOS processor. Specification of Processor is mentioned in Chapter 7. RTL design of Processor is briefed in Chapter 8. Chapter 9 shows the results and analysis. Chapter 10 discusses the summary and future scope. Finally there are two appendices at the end of the thesis report. Appendix-A focuses on the attributes of RISC /CISC processor. Appendix B represents an Assembly language program to copy the block of data from memory to memory.

CHAPTER 2

RISC (REDUCED INSTRUCTION SET COMPUTER)

2.1 Introduction

(Reduced instruction set computer) is a design philosophy aimed at delivering simple but powerful instructions that execute within a single cycle at high clock speed. The RISC philosophy concentrates on reducing the complexity of instructions performed by the hardware because it is easier to provide greater flexibility and intelligence in software rather than hardware. As a result, a RISC design places greater demands on compiler.

2.2 CISC Vs RISC

The traditional CISC (complex instruction set computer) relies more on the hardware for instruction functionality, and consequently the CISC instructions are more complicated. Figure 2 illustrates then difference between CISC and RISC

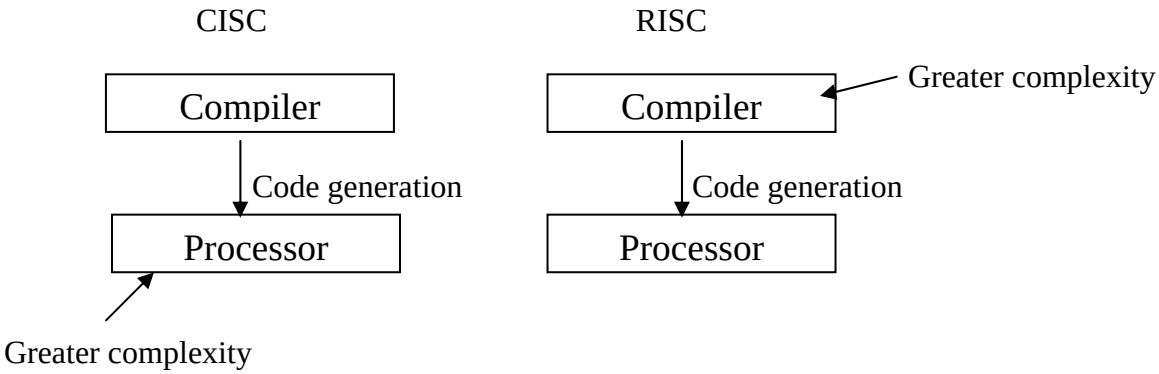

Figure 2.1 CISC vs. RISC

2.3 Design Rules for RISC processor

The RISC philosophy is implemented with four major design rules:

1. Instructions:

RISC processors have a reduced number of instruction classes. These classes provide simple operation that can each execute in a single cycle. The compiler or programmer

synthesizes complicated operations (for example, a divide instruction) by combining several simple instructions. Each instruction is a fixed length to allow pipeline to fetch future instructions before decoding the current instruction. In contrast, in CISC processors the instructions are often of variable size and take many cycles to execute.

2. Pipelines:

The processing of instructions is broken down into smaller units that can be executed in parallel by pipelines ideally the pipeline advances by one step on each cycle for maximize throughput. Instructions can be decoded in one pipeline stage. There is no need for an instruction to be executed by a miniprogram called microcode as on CISC processors.

3. Registers:

RISC machines have large general-purpose register set. Any register can contain either data or an address. Registers act as the fast local memory store for all data processing operations. In contrast, CISC processors have dedicated registers for specific purposes.

4. Load-Store architecture:

The processor operates on a data held in registers. Separate load and store instructions transfer data between the register bank and external memory. Memory accesses are costly, so separating memory accesses from data processing provide an advantage because you can use a data items held in register bank multiple times without needing multiple memory access. In contrast, with CISC design the data processing operations can act on memory directly.

CHAPTER 3

ARM PROCESSOR

3.1 Introduction

The ARM7TDMI* is a member of the Advanced RISC Machines (ARM) family of general-purpose 32-bit microprocessors, which offer high performance for very low power consumption and price. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles, and the instruction set and related decode mechanism are much simpler than those of micro programmed Complex Instruction Set Computers. This simplicity results in a high instruction throughput and impressive real-time interrupt response from a small and cost-effective chip. Pipelining is employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory. The ARM memory interface has been designed to allow the performance potential to be realized without incurring high costs in the memory system. Speed-critical control signals are pipelined to allow system control functions to be implemented in standard low-power logic, and these control signals facilitate the exploitation of the fast local access modes offered by industry standard dynamic RAMs.

A three-stage pipeline occupies minimal silicon area yet allows division of the execution time of each instruction into three parts: instruction fetch from memory, instruction decode, and instruction execution. The instruction execution stage is the most complex. Register read, a shift applied to one operand, an ALU operation, and finally a register writes all execute in one clock cycle. This limits the processor’s maximum clock speed to around 80 MHz on a 0.35-micron silicon process.

* T-Thumb
D-Debug
M-Multiplier
I-ICE

ARM7TDMI-S(Synthesizable) soft IP processor top-level block diagram is shown in Figure_3.1

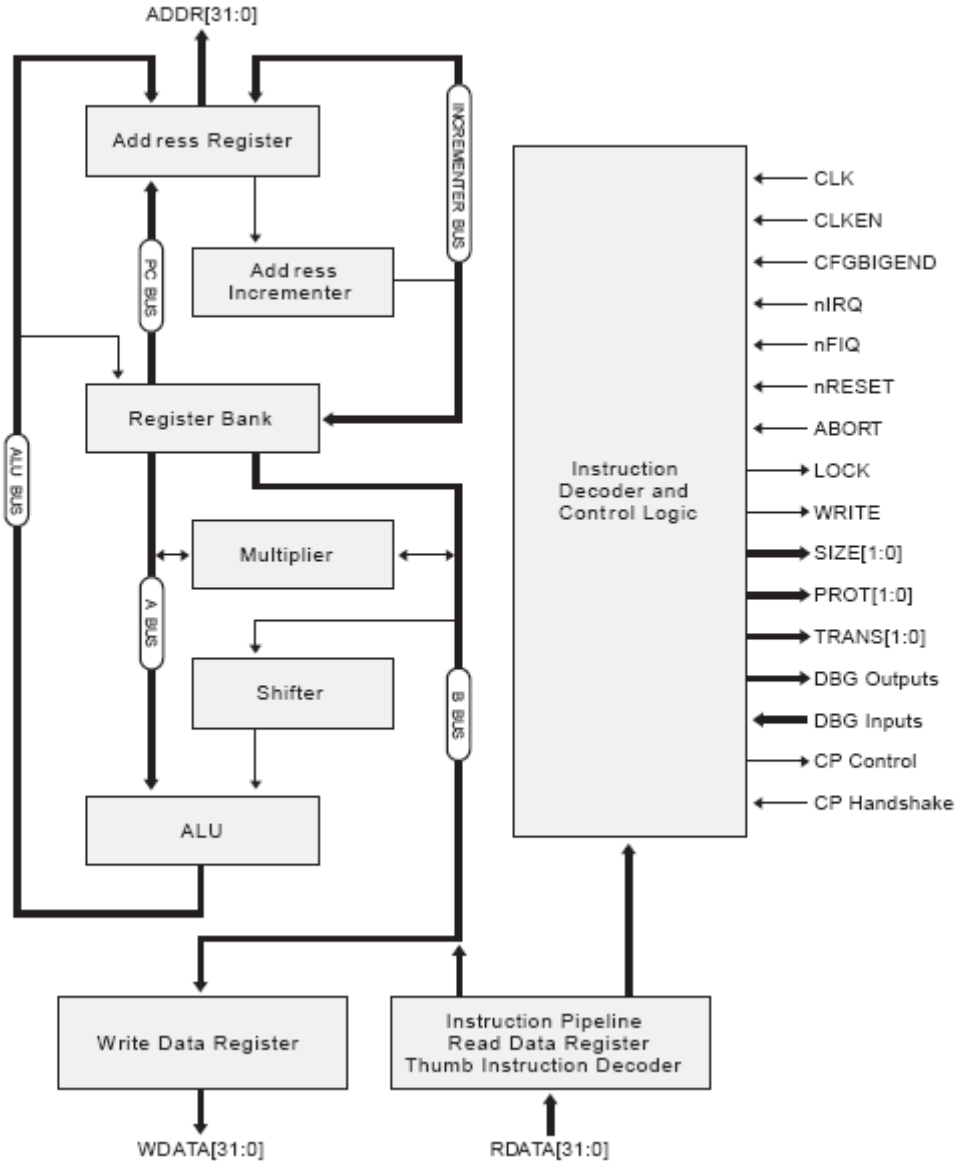

Figure 3.1 ARM7TDI Top-level block diagram

However, that speed is more than enough for the cost-sensitive applications using ARM7. The combined shift and ALU execution stage is also an important ARM feature. A single instruction can specify one of its two source operands for shifting or rotation before it is passed to the ALU. This allows very efficient bit manipulation and scaling code, and

virtually eliminates single shift instructions from ARM code. (The ARM processor does not have explicit shift instructions; a move instruction applies a shift to its operand.)

ARM7 also uses Von Neumann memory architecture; the instructions and data occupy a single address space and are accessed with individual address and data buses. Though this limits performance—instruction fetching (and hence execution) must stop for instructions that access memory—the reduced cost of a single memory outweighs performance in many embedded applications.

To reduce the penalty of data accesses stalling the pipeline, ARM implements load multiple and store multiple instructions. These instructions can move any of the ARM registers to and from memory, and update the memory address register automatically after the transfer. This not only allows one instruction to transfer many words of data (in a single bus burst), it also reduces the amount of instructions needed to transfer data. As a result, ARM code is smaller than other 32-bit instruction sets.

Although the pipeline stalls during load and store operations, the ARM7 can continue useful work. These instructions can specify an update of the base address register with a new address after (or even before) the transfer. RISC architectures would normally use a second instruction (add or subtract) to form the next address in a sequence. ARM does it automatically with a single bit in the instruction, again a useful saving in code size.

The ARM instruction set has one further useful feature. Most architecture has conditional branch instructions. These follow a test or compare instruction to control the flow of execution through the program. Some architecture also has a conditional move instruction, allowing data to be conditionally transferred between registers. The ARM instruction set takes this functionality to its logical extreme, allowing all instructions to be conditionally executed. Loads, stores, procedure calls and returns, and all other operations may execute conditionally after some prior instruction to set the condition code flags. (Any ALU instruction may set the flags.) This eliminates short forward branches in ARM code. Once again, this improves code density and avoids flushing the pipeline for branches, increasing execution performance.

ARM8 is the next core in the ARM line. It extends the ARM7 implementation in two fundamental ways: two additional pipeline stages and a new cache interface. ARM7's execute stage splits into three separate stages on ARM8, and register read moves back into the decode stage. The two additional pipeline stages perform memory accesses and register writes. Because each instruction executes over multiple cycles, register-forwarding paths must pass data between successive instructions. This is necessary because one instruction will not have written its result to the register file before the next two instructions have read their source register values. ARM8 incorporates a single cache interface that allows instruction fetches in parallel with data accesses. It retains ARM7's Von Neumann cache interface, but doubles the bandwidth of the interface to provide 64 bits every cycle. ARM8 also uses a sophisticated prefetch buffer and branch prediction unit to fetch instructions ahead of the execution unit. On every cycle, one instruction is fed to the processor from the prefetch buffer. When the cache is not in use for a data access, two instructions are loaded into the prefetch buffer. This allows the single cache to satisfy both data and instruction accesses.

ARM8 behaves similarly in performance to a Harvard machine with separate instruction and data caches, yet retains the simplicity of a single cache machine. Static branch prediction predicts the target of branch instructions; backward branches are assumed taken (loops) and forward branches untaken (conditional code). Correctly predicted branches do not enter the main execution engine and thus effectively execute in zero cycles. Mispredicted branches take three cycles to correct. ARM8 delivers 100- MHz operation in a typical 0.35-micron process, and lowers the average number of clock ticks per instruction to around 1.5. This increases overall performance by about 70% over ARM7. Digital Equipment Corporation code designed the StrongARM1, the fastest of our current processors. Adoption of Harvard architecture to deliver maximum cache throughput and a five-stage instruction pipeline to allow maximum clock rate produced an embedded processor that is faster than some workstation processors. StrongARM110 incorporates two 16-Kbyte caches maintained even when the processor is coupled to a relatively low-speed memory system. When coupled with Digital's very fast 0.35-micron process, which operates with a 2-volt supply, StrongARM1 machines deliver 233 MHz.

With less than 1 Watt of power consumption, this makes the StrongARM power consumption/performance ratio the best in the industry.

3.2 ARM7 register set

The ARM processor, like all RISC processors, uses load-store architecture. This means it has two instruction types for transferring data in and out of processor: load instructions copy data from memory to registers in the core, and conversely the store instructions copy data from registers to memory. There are no data processing instructions that directly manipulate data in memory. Thus, data processing is carried out solely in registers.

Data items are placed in the register file- a storage bank made up of 32-bit registers.

• Register structure depends on mode of operation
• 16 pieces of 32-bit integer registers R0 - R15 are available in ARM-mode (usr, user)
• R0 - R12 are general-purpose registers
• R13 is Stack Pointer (SP)
• R14 is subroutine Link Register
• Holds the value of R15 when BL-instruction is executed
• R15 is Program Counter (PC)
• Bits 1 and 0 are zeroes in ARM-state (32-bit addressing)
• R16 is state register (CPSR, Current Program Status Register)

3.3 Processor modes

The processor mode determines which registers are active and the access rights to the cpsr register itself. Each processor mode is either privileged or non privileged: a privileged mode allows full read-write access to the CPSR. Conversely, a nonprivelleged mode only allows read access to the control field in the cpsr but still allows read-write access to the control flags. There are seven processor modes in total: six privileged modes and one nonprivileged mode as mentioned below

Nonprivileged mode

• User mode: Normal program execution state

Privileged mode

• FIQ (fast interrupt request) mode: Data transfer state (fast irq, DMA-type transfer)
• IRQ (interrupt request) mode: Used for general interrupt services
• Supervisor (svc) mode: Protected mode for operating system support
• Abort mode (abt) mode: Selected when data or instruction fetch is aborted
• System (sys) mode: Operating system 'privilege'-mode for user
• Undefined (und) mode: Selected when undefined instruction is fetched

The processor enters abort mode when there is failed attempt to access memory. Fast interrupt request and interrupt request modes correspond to the two-interrupt levels available on the ARM processor. Supervisor mode is the mode that the processor is in after reset and is generally the mode that an operating system kernel operates in. Undefined mode is used when the processor encounters an instruction that is undefined or not supported by the implementation. User mode is used for programs and applications.

3.4 Instruction set

Arm architecture supports 32- bit ARM instruction set and 16- bit THUMB instruction set.

It supports different types of instructions:

- Data processing
- Arithmetic
- Multiplication
- Logical
- Comparison
- Shift rotate
- Branch
- Exception

- Load/ store instruction
- Co-processor instruction

• In normal instruction execution (unconditional) condition field contents of AL is used (Always)
• In conditional operations one of the 14 available conditions is selected
• For example, instruction known usually as BNZ in ARM is NE (Z-flag clear) conditioned branch-instruction

3.5 ARM7TDMI Architecture

The ARM7TDMI processor employs a unique architectural strategy known as THUMB, which makes it ideally suited to high-volume applications with memory restrictions, or applications where code density is an issue.

3.6 The THUMB Concept

The key idea behind THUMB is that of a super-reduced instruction set. Essentially, the ARM7TDMI processor has two instruction sets:

• The standard 32-bit ARM set
• A 16-bit THUMB set

The THUMB set's 16-bit instruction length allows it to approach twice the density of standard ARM code while retaining most of the ARM's performance advantage over a traditional 16-bit processor using 16-bit registers. This is possible because THUMB code operates on the same 32-bit register set as ARM code. THUMB code is able to provide up to 65% of the code size of ARM, and 160% of the performance of an equivalent ARM processor connected to a 16-bit memory system.

THUMB's Advantages

THUMB instructions operate with the standard ARM register configuration, allowing excellent interoperability between ARM and THUMB states. Each 16-bit THUMB

instruction has a corresponding 32-bit ARM instruction with the same effect on the processor model.

The major advantage of a 32-bit (ARM) architecture over a 16-bit architecture is its ability to manipulate 32-bit integers with single instructions, and to address a large address space efficiently. When processing 32-bit data, a 16-bit architecture will take at least two instructions to perform the same task as a single ARM instruction. However, not all the code in a program will process 32-bit data (for example, code that performs character string handling), and some instructions, like Branches, do not process any data at all. If a 16-bit architecture only has 16-bit instructions, and a 32-bit architecture only has 32-bit instructions, then overall the 16-bit architecture will have better code density, and better than one half the performance of the 32-bit architecture. Clearly 32-bit performance comes at the cost of code density. THUMB breaks this constraint by implementing a 16-bit instruction length on a 32-bit architecture, making the processing of 32-bit data efficient with a compact instruction coding. This provides far better performance than a 16-bit architecture, with better code density than a 32-bit architecture. THUMB also has a major advantage over other 32-bit architectures with 16-bit instructions. This is the ability to switch back to full ARM code and execute at full speed. Thus critical loops for applications such as

- Fast interrupts
- DSP algorithms

can be coded using the full ARM instruction set, and linked with THUMB code. The overhead of switching from THUMB code to ARM code is folded into sub-routine entry time. Various portions of a system can be optimized for speed or for code density by switching between THUMB and ARM execution as appropriate.

3.7 Exception and interrupt handling

At the heart of an embedded system lie the exception handlers. They are responsible for handling errors, interrupts, and other events generated by the external system. Efficient handlers can dramatically improve system performance. The process of determining a good handling method can be complicated and challenging.

An exception changes the normal sequential execution of instructions. The ARM processor has seven exceptions that can halt the normal sequential executions: Data Abort, FIQ, IRQ, Prefetch Abort, Software Interrupt, Reset, and Undefined Instruction. Each exception has an associated ARM processor mode. When exception is raised, the processor goes into a specific mode and branches to an entry in the vector table. Each exception also has a priority level.

Interrupts are special type of exception that are caused by an external peripheral. The IRQ exception is used for general operating system activities. The FIQ exception is normally reserved for a single interrupt source.

3.8 Using FPGAs with ARM Processors

FPGAs are known for providing designers with several benefits in system design. One of the most important has been lessening the time to market. The quicker a company gets its products to market, the more market share it can capture from its competitors. This could mean millions of dollars in income to an established company and make or break a young company. Another major benefit that FPGAs provide is flexibility. Designers can modify their design up to the day that the product is released to customers. And now with the concept of Internet Reconfigurable Logic (IRL), designs can be modified even after they are shipped to customers. More recently, FPGAs have become attractive for other reasons. Because the FPGA cost per gate has come down significantly, there is no longer a ‘price penalty’ associated with the benefits of programmable logic. FPGAs have become an attractive option in many high volume applications. FPGA density having dramatically increased, Xilinx is now shipping multi-million gate devices. Certainly, this number will continue to climb in the future. This trend not only allows the design engineer to consider programmable logic for larger designs, but also allows them to absorb the functionality of other on-board chips. Fewer chips mean lower cost and easier board layout. And finally, there is the benefit of performance. Application Specific Integrated circuits (ASICs) built with cutting edge technology will always be faster, but FPGAs are close behind. Along with better on-chip performance, the I/O performance in Xilinx FPGAs has improved. One of the ways used to improve I/O performance and provide more flexibility was the introduction of Select I/O, which gives the designer the

ability to choose an I/O standard suited of his system design. With the Xilinx FPGA's flexible Select I/O feature, programmable logic chips can now interface with almost any other component. Xilinx has already documented how to interface with an array of memories and peripheral busses. Xilinx has created the Memory Corner as a one-stop memory shop, providing solutions for leading edge memory technology.

Another component that is found in almost every system is the microprocessor, which comes in many flavors to suit system needs. The ARM microprocessor has gained popularity because of its features, peripherals, low power, and flexibility.

3.9 ARM as a standard component

Even tough ARM is mostly used as a processor core in SoC and other ASICs have some manufacturers brought ARM based standard products to market. Examples of manufacturers are Atmel, Cirrus Logic, Hyundai, Intel, Oki, Samsung and Sharp. Most of the products are based on 7TDMI-core, some to 720Tand 920T-cores.In addition, there are a number of ASSP (Application Specific Standard Product) -chips available for example to communication applications (Philips VWS22100 = ARM7 based GSM base band chip) using ARM processor as a core.

3. 10 Tool set of ARM processor

3.11 Applications of ARM processors

ARM processors are found in numerous market segments, including networking, automotive, mobile and consumer devices, mass storage, and imaging. Within each segment ARM processors can be found in multiple applications.

For example, the ARM processor is found in networking applications like home gateways, DSL modems for high speed Internet communication, and 802.11 wireless communications. The mobile device segment is the largest application area for ARM processors because of mobile phones. ARM processors are also found in mass storage devices such as hard drives and imaging products such as inkjet printers-applications that are cost sensitive and high volume.

In contrast, ARM processors are not found in applications that require leading-edge high performance. Because these applications tend to be low volume and high cost, ARM has decided not to focus designs on these types of applications**.**

CHAPTER 4

NIOS_II PROCESSOR

4.1 Introduction

A Nios II processor system is equivalent to a microcontroller or "computer on a chip" that includes a CPU and a combination of peripherals and memory on a single chip. The term "Nios II processor system" refers to a Nios II processor core, a set of on-chip peripherals, on chip memory, and interfaces to off-chip memory, all implemented on a single Altera chip. Like a microcontroller family, all Nios II processor systems use a consistent instruction set and programming model.

4.2 Nios II Processor System Basics

The Nios II processor is a general-purpose RISC processor core, providing:

- Full 32-bit instruction set, data path, and address space
- 32 general-purpose registers
- 32 external interrupt sources
- Single-instruction 32 *32 multiply and divide producing a 32-bit result
- Dedicated instructions for computing 64-bit and 128-bit products of multiplication
- Floating-point instructions for single-precision floating-point operations
- Single-instruction barrel shifter
- Access to a variety of on-chip peripherals, and interfaces to off-chip memories and peripherals
- Hardware-assisted debug module enabling processor start, stop, step and trace under integrated development environment (IDE) control
- Software development environment based on the GNU C/C++ tool chain and Eclipse IDE

- Integration with Altera's Signal Tap(r) II logic analyzer, enabling real time analysis of instructions and data along with other signals in the FPGA design
- Instruction set architecture (ISA) compatible across all Nios II processor systems
- Performance up to 250 DMIPS

4.3 Performance Parameters (NIOS_II)

These parameters are the actual performance parameters of the system, which were obtained after implementing the system design. The actual system performance parameters are:

Power supply—DC voltage: 9 – 12 V; operating current: 600 mA (motherboard current is about 200 mA); power consumption: 10 W

Environment—operating temperature: 0° C – 40° C; relative humidity: 8% – 95%

Input image—Mode: two-channel asynchronous image; data format after digitalization: CCIR-656

Display—LCD resolution: 640 x 480; LCD display area: 16 cm^2 or 6.4 $inch^2$; LCD display color depth: 6-bit/RGB

Storage—Storage image resolution: 640 x 480; storage image color depth: 8-bit, JPEG; storage image capacity: about 100 Kbytes/image; image storage total cycle: < 0.5 s/2 images

Transmission time—about 1 minute/image (depending on local network condition)

4.4 Configurable Soft-Core Processor

The Nios II processor is a configurable soft-core processor, as opposed to a fixed, off-the-shelf micro controller. In this context, “configurable” means that features can be added or removed on a system-by-system basis to meet performance or price goals. “Soft-core” means the CPU core is offered in “soft” design form (i.e., not fixed in silicon), and can be targeted to any Altera FPGA family. In other words, Altera does not sell “Nios II chips”; Altera sells blank FPGAs. It is the users that configure the Nios II processor and

peripherals to meet their specifications, and then program the system into an Altera FPGA. Configurability does not mean that designers must create a new Nios II processor configuration for every new design. Altera provides readymade Nios II system designs that system designers can use as-is. If these designs meet the system requirements, there is no need to configure the design further. In addition, software designers can use the Nios II instruction set simulator to begin writing and debugging Nios II applications before the final hardware configuration is determined.

4.5 Designing with Nios II and SOPC Builder

SOPC Builder is a powerful system development tool for creating systems based on processors, peripherals, and memories. SOPC Builder enables you to define and generate a complete system-on-a-programmable-chip (SOPC) in much less time than using traditional, manual integration methods. SOPC Builder is included in the Quartus II software and is available to all Altera customers. Many designers already know SOPC Builder as the tool for creating systems based on the Nios® II processor. However, SOPC Builder is more than a Nios II system builder; it is a general-purpose tool for creating arbitrary SOPC designs that may or may not contain a processor. SOPC Builder automates the task of integrating hardware components into a larger system. Using traditional system-on-chip (SOC) design methods, you had to manually write top-level HDL files that wire together the pieces of the system. Using SOPC Builder, you specify the system components in a graphical user interface (GUI), and SOPC Builder generates the interconnect logic automatically. SOPC Builder outputs HDL files that define all components of the system, and a top-level HDL design file that connects all the components together. SOPC Builder generates both Verilog HDL and VHDL equally, and does not favor one over the other.

In addition to its role as a hardware generation tool, SOPC Builder also serves as the starting point for system simulation and embedded software creation. SOPC Builder provides features to ease writing software and to accelerate system simulation.

An SOPC Builder *component* is a design module that SOPC Builder recognizes and can automatically integrate into a system. SOPC Builder connects multiple components together to create a top-level HDL file called the *system module*.

SOPC Builder components are the building blocks of the system module. SOPC Builder components use the Avalon interface for the physical connection of components, and you can use SOPC Builder to connect any logical device (either on-chip or off-chip) that has an Avalon interface.

The Avalon interface uses an address-mapped read/write protocol that allows master components to read and/or write any slave component.

Altera and third-party developers provide ready-to-use SOPC Builder components, such as:

- Microprocessors, such as the Nios II processor
- Micro controller peripherals
- Timers
- Serial communication interfaces, such as a UART and a serial peripheral interface (SPI)
- General purpose I/O
- Digital signal processing (DSP) functions
- Communications peripherals
- Interfaces to off-chip devices
    - Memory controllers
    - Buses and bridges
    - Application-specific standard products (ASSP)
    - Application-specific integrated circuits (ASIC)
    - Processors

4.6 Custom instruction

With the Altera Nios_II embedded processor, system designers can accelerate time-critical software algorithms by adding custom instructions to the Nios instruction set.

With custom instructions, system designers can reduce a complex sequence of standard instructions to a single instruction implemented in hardware. System designers can use this feature for a variety of applications, e.g., to optimize software inner loops for digital signal processing (DSP), packet header processing, and computation-intensive applications. The Nios II CPU configuration wizard, which is accessed via the Quartus_II software's SOPC Builder, provides a graphical user interface (GUI) used to add up to 256 custom instructions to the Nios II processor.

The custom instruction logic connects directly to the Nios II arithmetic logic unit (ALU) as shown in Figure 4.1

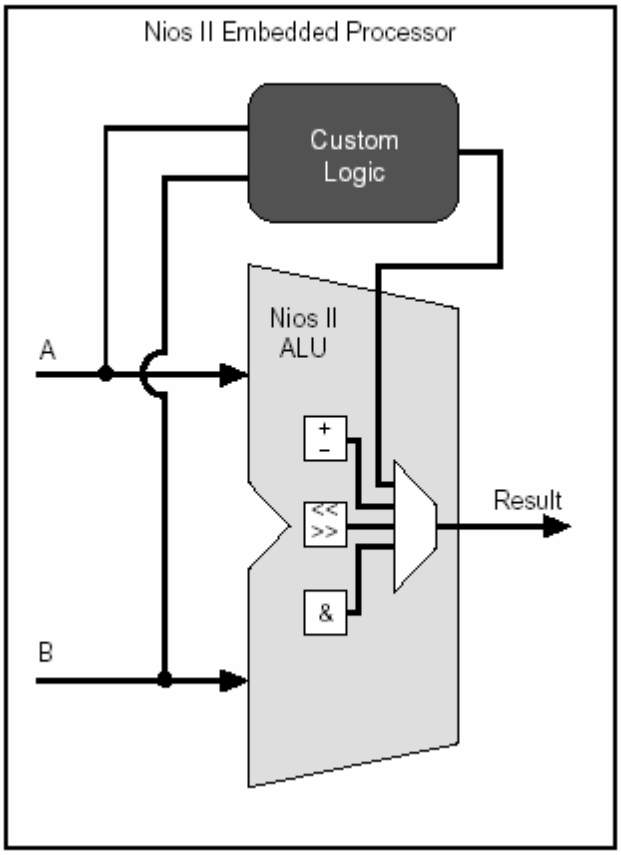

Figure 4.1 Custom instruction logic

With Nios II processor custom instructions, system designers are able to take full advantage of the flexibility of FPGAs to meet system performance requirements. Custom

instructions allow system designers to add custom functionality to the Nios II processor ALU.

Nios II processor custom instructions are custom logic blocks adjacent to the ALU in the CPU's data path. This gives system designers the ability to tailor the Nios II processor core to meet the needs of a particular application. System designers have the ability to accelerate time critical software algorithms by converting them to custom hardware logic blocks. Because it is easy to alter the design of the FPGA-based Nios II processor, custom instructions provide an easy way to experiment with hardware/software trade-offs during an embedded system's implementation phase—rather than the specification phase.

4.7 Custom Instruction Architecture

Combinatorial custom instruction architecture consists of a logic block that is able to complete in a single clock cycle.

Figure 4.2 shows a block diagram of combinatorial custom instruction architecture.

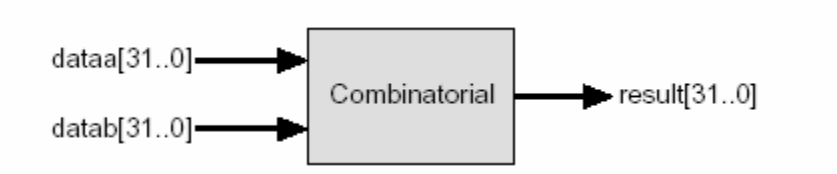

Figure 4.2 Combinatorial Custom Logic

The figure combinatorial custom instruction diagram uses data a [31..0] and data b[31..0] ports as inputs and drives the results on the result [31..0] port. Because the logic is able to complete in a single clock cycle, control signals are not needed.

Figure 4.3 shows a simple, multiply-accumulate custom logic block.

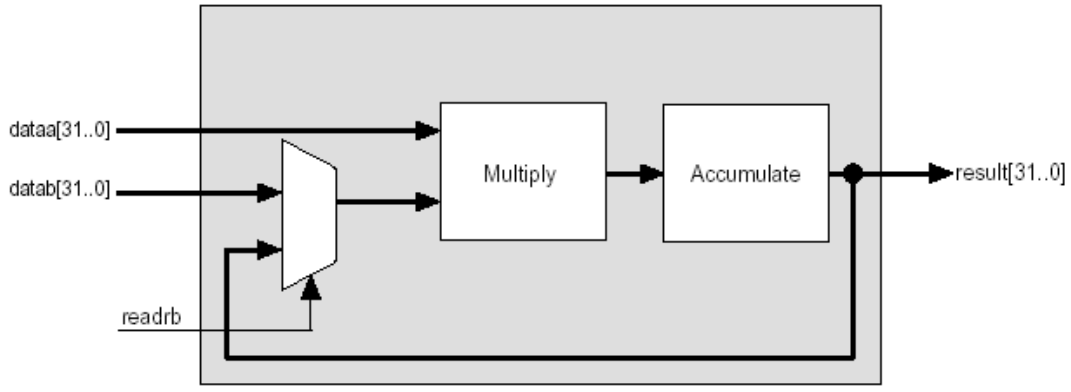

Figure 4.3 Multiply-Accumulate custom logic block

CHAPTER 5
POWERPC PROCESSOR

5.1 Introduction

PowerPC is a microprocessor RISC-based instruction set architecture (ISA) developed in 1991 by IBM, Motorola (now Freescale Semiconductor), and Apple Computer. A new version of the PowerPC ISA will be released in August 2006 by the Power.org™ Power Architecture™ Advisory Council (PAAC). This new version—Power ISA™ 2.03—is a component of the Power Architecture platform, which also consists of a broad community of supporters, software and tools, and products that are built on the Power Architecture platform. Given its wide use, PowerPC version 1.x will continue to be the basis of some products in the marketplace.

The PPC405 is a 32-bit implementation of the PowerPC embedded-environment architecture that is derived from the PowerPC architecture. Specifically, the PPC405 is an embedded PowerPC 405D5 processor core.

The PowerPC architecture provides a software model that ensures compatibility between implementations of the PowerPC family of microprocessors. The PowerPC architecture defines parameters that guarantee compatible processor implementations at the application-program level, allowing broad flexibility in the development of derivative PowerPC implementations that meet specific market requirements.

5.2 PPC405 Features

The PPC405 processor core is an implementation of the PowerPC embedded-environment architecture. The processor provides fixed-point embedded applications with high performance at low power consumption. It is compatible with the PowerPC UISA.

Key features of the PPC405 include:

- Embedded PowerPC 405 (PPC405) core
- Embedded 400 MHz, 600+ DMIPS RISC core (32-bit Harvard architecture)
- 5-stage data path pipeline
- Hardware multiply and divide
- 32 x 32-bit general-purpose registers
- 16 KB 2-way set-associative instruction and data caches
- Memory Management Unit (MMU) enables RTOS implementation
- Flexible memory management
- 64-entry unified Translation Look-aside Buffers (TLB)
- Variable page sizes (1KB - 16 KB)
- Enhanced instruction and data On-Chip Memory (OCM) controllers interface directly to embedded Block RAM
- Supports IBM Core Connect bus architecture
- Debug and trace support
- Advanced power management support
- Minimized interrupt latency
- Timer facilities -programmable interval timer (PIT), fixed interval timer (FIT), and watchdog timer (All are synchronous with the time base)

5.3 PowerPC Architecture Overview

The PowerPC architecture is a 64-bit architecture with a 32-bit subset. In general, the PowerPC architecture defines the following:

- Instruction set
- Programming model
- Memory model
- Exception model
- Memory-management model
- Time-keeping model

Instruction Set

The instruction set specifies the types of instructions (such as load/store, integer arithmetic, and branch instructions), the specific instructions, and the encoding used for the instructions. The instruction set definition also specifies the addressing modes used for accessing memory.

Programming Model

The programming model defines the register set and the memory conventions, including details regarding the bit and byte ordering, and the conventions for how data are stored.

Memory Model

The memory model defines the address-space size and how it is subdivided into pages. It also defines attributes for specifying memory-region cacheability, byte ordering (big-endian or little endian), coherency, and protection.

Exception Model

The exception model defines the set of exceptions and the conditions that can cause those exceptions. The model specifies exception characteristics, such as whether they are precise or imprecise, synchronous or asynchronous, and maskable or non-maskable. The model defines the exception vectors and a set of registers used when interrupts occur as a result of an exception. The model also provides memory space for implementation-specific exceptions.

Memory-Management Model

The memory-management model defines how memory is partitioned, configured, and protected. The model also specifies how memory translation is performed, defines special memory-control instructions, and specifies other memory-management characteristics.

Time-Keeping Model

The time-keeping model defines resources that permit the time of day to be determined and the resources and mechanisms required for supporting timer-related exception

5.4 PPC405 Organization

As shown in Figure 5.1 the PPC405 processor contains the following elements:

- A 5-stage pipeline consisting of fetch, decode, execute, write-back, and load write back stages
- A virtual-memory-management unit that supports multiple page sizes and a variety of storage-protection attributes and access-control options
- Separate instruction-cache and data-cache units
- Debug support, including a JTAG interface
- Three programmable timers

The following sections provide an overview of each element.

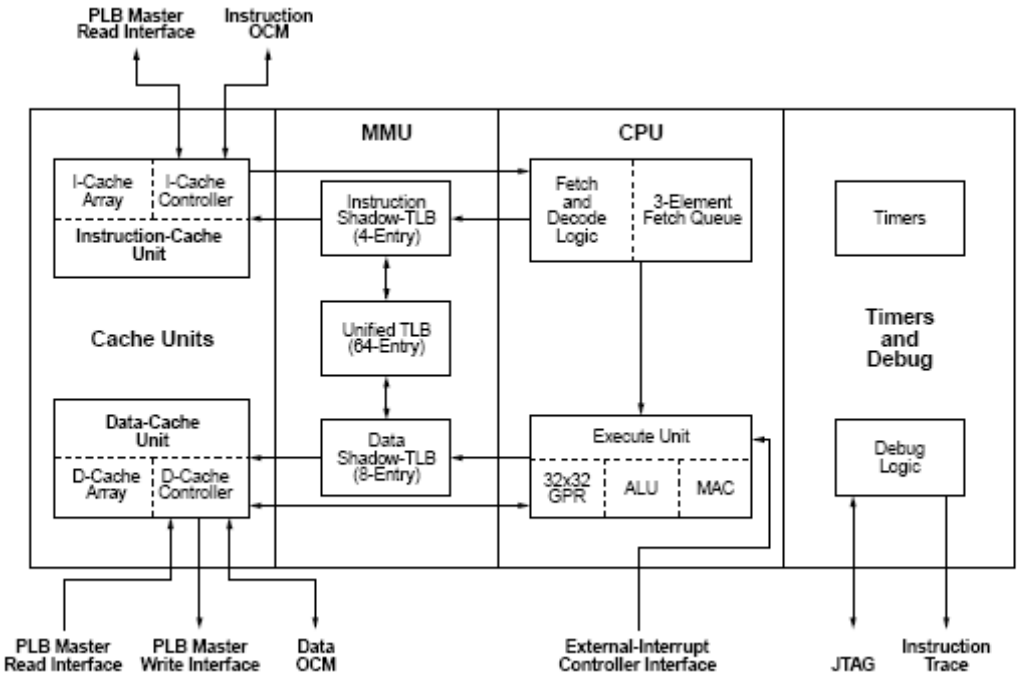

Figure 5.1 PPC405 Organization

5.5 PowerPC Embedded-Environment Architecture

The PowerPC embedded-environment architecture is optimized for embedded controllers. This architecture is a forerunner to the PowerPC Book-E architecture. The

PowerPC embedded-environment architecture provides an alternative definition for certain features specified by the PowerPC VEA and OIA. Implementations that adhere to the PowerPC embedded-environment architecture also adhere to the PowerPC UISA. PowerPC embedded-environment processors are 32-bit only implementations and thus do not include the special 64-bit extensions to the PowerPC UISA. Also, floating-point support can be provided either in hardware or software by PowerPC embedded-environment processors.

5.6 Timer Resources

The PPC405 supports several timer resources that can be used for a variety of time-keeping functions. Possible uses of these timer resources include:

- Time-of-day computation.
- Data-logging for system-service routines.
- Periodic servicing of time-sensitive external devices.
- Preemptive multitasking.

The timer resources supported by the PPC405 consist of:

• Two timer registers:

- A 64-bit incrementing timer called the time-base.
- A 32-bit decrementing timer called the programmable-interval timer.

• Three timer-event interrupts:

- A watchdog-timer interrupt that provides the ability to set critical interrupts that can aid in recovery from system failures.
- A programmable-interval timer interrupt that provides the ability to set non critical variable-time interrupts.
- A fixed-interval timer interrupt that provides the ability to set non-critical interrupts with a fixed, repeatable time period.

• A timer-control register for setting up and controlling the timer events.

• A timer-status register for recording timer-event status.

5.7 Tool set of PowerPC processor

CHAPTER 6

COMPARISION OF ARM PROCESSOR AND NIOS II PROCESSOR

Based on the study of ARM processor and NIOS_II processor the summary of features are listed out and shown in Table 6.1

Table 6.1 Summary of features of ARM processors and NIOS_II processor

Sr. no	Parameter	Nios_II	Arm7TDMI
1.	Type of processor	RISC	RISC
2.	Size of data bus	32 bit	32 bit
3.	Size of address bus	32 bit	32 bit
4.	No. of general-purpose registers	32# of 32 bit size	31# of 32 bit size (16 are visible)
5.	No. of control registers	06# of 32 bit size	06# of 32 bit size
6.	Form of processor	Soft processor (to be implemented on FPGA)	In form of soft processor as well as hard processor
7.	Type of architecture	Havard type (Separate instruction bus and Data Bus)	Von Neumann type
8.	Exception support	32 external interrupt sources	Interrupt controller is required
9.	Exception Types	1**. Hardware interrupt** (32#) 2. **Software interrupt**- TRAP instruction 3 **Unimplemented instruction**- Mul, muli, mulxss, mulxsu, Mulxuu, div, divu	1. Fast interrupt 2. Normal interrupt 3. Memory aborts 4. Attempted execution of undefined instruction

		4. **Other**-defined at the time of publishing e.g. for MMU	5. Software interrupt (SWI-instruction)
10.	Instruction customization	Possible (example-MAC)	Not possible (MACRO is possible)
11.	Nature of memory and I/O organization	As NIOS-II is configurable memory and I/O mapping is done at system generation system time.	
12.	I/O organization	Memory-mapped -32 bit. Same space is used for memory and I/O	Memory-mapped -32 bit. Same space is used for memory and I/O
13.	Software development environment/ Cross compiler support	C/C++	Armasm assembler
14.	Performance	31 DMIPS (nios_II/e) 127 DMIPS (nios_II/s) 218 DMIPS (nios_II/f)	0.97 MIPS/MHz
15.	Reset Signals	Global and cpu_reset	Global and cpu_reset
16.	Cache memory	For both data and instruction	For both data and instruction
17.	Cache memory size (on chip)	512 bytes to 64 Kbytes	512 bytes to 96 Kbytes
18.	Debug	Using JTAG	Using JTAG
19.	Mode of operation	1. User mode 2. Supervisor mode	1. User mode 3. Supervisor mode 4. System mode 5. Interrupt request

			6. Fast interrupt 7. Abort mode 8. Undefined mode
20.	Addressing Modes	1. Register addressing 2. Displacement addressing 3. Immediate addressing 4. Register indirect addressing 5. Absolute addressing	1. data processing 2. load and store coprocessor 3. load store multiple 4. load and store word or unsigned byte miscellaneous loads/ stores
21.	Instruction Set	32- bit instruction set	32- bit ARM instruction set 16- bit THUMB instruction set
22.	Instruction Set category	1. Data Transfer -LDW- load from memory 2. Arithmetic & Logical – XOR, SUB 3. Move-MOVI 4. Comparison -CMPGT 5. Shift & Rotate-ROL, SLL 6. Program Control- CALL, JMP 7. Other Control Instructions-TRAP,	1. Data processing 2. Arithmetic 3. Multiplication 4. Logical 5. Comparison 6. Shift rotate 7. Branch 8. Exception 9. Load/ store instruction 10. Co-processor

		BREAK 8. Custom Instructions-can be implemented 9. No Operation-NOP	instruction
23.	Device Support	Altera FPGAs 1. Stratix® 2. Stratix II 3. CycloneTM 4. Cyclone II	Actel FPGAs
24.	No. of pipe line stages	Six stage (nios_II/f) Fetch-Decode-Execute-Memory-Align-Write back	3 stage Fetch-Decode-Execute
25.	Instruction word format	Three categories: 1. I-type- 6 bit opcode field Two 5-bit reg, fields 16-bit immediate data Exe. ADDI RB RA 16- BIT DATA 2. R-type, and 6 bit opcode field three 5-bit reg, fields 11-bit opcode extension field Exe. ADD RB RA RC 3. J-type. 6 bit opcode field 26-bit immediate data Exe CALL label	32-bit format (no various cateqories)
26.	Max frequency of	200 MHz (nios_II/e)	60-110 MHz

	operation: Fmax	165 MHz (nios_II/s) 185MHz (nios_II/f)	
27.	Power Consumption	System specification stratix based - DC voltage: 9 – 12 V; operating current: 600 mA (motherboard current is about 200 mA);	Voltage- 1.8 V Power consumption-.25mW
28.	Parallelism in FPGA…?How..? Multiprocesor support in HW(Max.. how many..In what configuration…Linear-mesh.- Matrix, Cube and SW-(C/C++ and IDE support)	Possible- multiprocessor design is possible	Possible-multiprocessor design is possible
29.	External memory addressing and size	Automated & 2GB	4 GB
30.	DMA Data transfer support	Yes	Yes

In order to get best RISC processor for DSP application, the studies of various processor cores such as ARM processor, NIOS_II processor, PowerPC. is carried out. The reason to study said cores is that these are available in form of soft core module so it can be down loaded to FPGA. We can modify the logic as per our application or we can do value addition

As a conclusion, based on study of various cores, I have following points

ARM processor is better compared to others in following aspects

1. barrel shifter

In line barrel shifter is a hardware component that preprocessed one of the input registers before it is used by an instruction. This expands the capability of many instructions to improve core performance and density

e.g Suppose R0=0x00000000

R1=0x00000005

ADD RO R1, R1, LSL #1

After execution of above instruction

The content of above two registers will be as below

R0=0x0000000F

R1=0x00000005

2. Load store multiple instructions

To transfer multiple registers between memory and the processor in a single instructions.

e.g to load three registers from memory

LDMIA R0!, {R1-R3}

R0 is the base address

IA -increment after (specify addressing mode)

Other addressing modes are IB, DA, DB

3. Thumb instruction set

That permits the ARM core to execute either 16 or 32 bit instructions. The 16-bit instructions improve code density by about 30% over 32-bit fixed length instructions.

e. g. To Divide the content of R0 by R1

- ARM CODE

```
        MOV R3, #0
LOOP1:  SUB R0, R0, R1
        ADDGE R3, R3, #1
        BGE LOOP1
        ADD R2, R0,R1
```

Code size: 5*4=20 bytes

THUMB CODE

```
        MOV R3, #0
LOOP2:  ADD R3, #1
        SUB R0, R1
        BGE LOOP2
        SUB R3, #1
        ADD R2, R0, R1
```

Code size: 6*2=12 bytes

3. conditional execution

   An instruction is only executed when specific condition has been satisfied. This feature improve performance and code density by reducing branch instructions

   e.g. ADDEQ R1, R2, #4

4. Enhanced instructions:

   a. Parallel arithmetic

   Half word or byte wise addition or substraction

   e. g. ADD16 R2, R3, R4

   SUB8 R3, R4, R5

   Half word or byte wise exchange, addition and substraction

   e. g. ADDSUBX R1, R3,R4

   b. The enhanced DSP instructions were added to standard Arm Instruction to support fast multiplier operation

   DSP instructions

   - Multiply, add and accumulate

     e. g. MLA R1, R2, R3, R4

Parallel arithmetic

- Half word or byte wise addition or subtraction

  e. g. ADD16 R2, R3, R4

  SUB8 R3, R4, R5

- Half word or byte wise exchange, addition and subtraction

  e.g ADDSUBX R1, R3,R4

5. Has count leading zero instructions

This instruction counts the no. of zeros between the MSB and first bit set to 1

CLZ R0, R1

About 75 % of core in embedded system is of ARM processor.

PowerPC is better compared to others in following regards

1. Performance

Processor	**Maximum frequency**	**Performance**
PowerPC	450Mhz	700 MIPS
NIOS_II	185 Mhz	218 MIPS
ARM	60-110 Mhz	0.97 MIPS/Mhz

Table 6.2 Processor Performance Evaluation

2. Timer facility

16-bit incremental count timer for three modes

a. watchdog timer

b. fixed interval timer

c. programmable timer

Nios_II is better compared to others in following

Custom Instruction

The soft-core nature of the Nios II processor lets designers integrate custom logic into the arithmetic logic unit (ALU). Similar to native Nios II instructions, custom instructions accept values from up to two 32-bit source registers and optionally write back a result to a 32-bit destination register. By using custom instructions, The system designers can fine-tune the system hardware to meet performance goals and also the designer can easily handle the instruction as a macro in C/C++.

Custom Peripherals

System designers also can create their own custom peripherals that can be integrated with Nios II processor systems. For performance-critical systems that spend most CPU cycles executing a specific section of code, it is a common technique to create a custom peripheral that implements the same function in hardware. Using this approach, performance is doubled: **the hardware implementation is faster than software**; and the processor is free to perform other functions in parallel while the custom peripheral operates on data.

Based on Instruction set strength, we can compare processors as below:

Parameter	ARM	NIOS_II	PowerPC
Size	32-bit & 16-bit	32 bit	32 bit
Code compactness	Possible using Thumb instruction	Not possible	Not possible
Custom instruction	Not possible	By adding custom logic	Not possible
Types of instructions	10	9	9
Special instructions	Parallel arithmetic CLZ	-	Timer based

Table 6.3 Instruction Set Evaluation

Processors are compared based on exceptions/interrupts are as follow

Parameter	**ARM**	**NIOS_II**	**PowerPC**
H/w interrupt	Support	Support	Support
Software interrupt	SWI	TRAP	TW
Other interrupts	Undefined instruction	Unimplemented instruction-	Timer out

Table 6.4 Exception Evaluation

Hence ARM processor is more suitable for DSP application. We can customize some instructions based on DSP application to get the best outcomes.

CHAPTER 7

RISC PROCESSOR SPECIFICATIONS

GENERAL SPECIFICATIONS OF RISC PROCESSOR

Sr. no	Parameter	Value
1.	Type of processor	RISC
2.	Size of data bus	32 bit
3.	Size of address bus	32 bit
4.	ALU	32 bit
5.	No. of general-purpose registers	32# of 32 bit size
6.	Form of processor	Soft processor (to be implemented on FPGA)
7.	I/O organization	Memory-mapped -32 bit. Same space is used for memory and I/O
8.	Reset Signals	Global and cpu reset
9.	Addressing Modes	All general purpose
10.	Instruction Set	32- bit instruction set
11.	Instruction Set category	All general purpose –data transfer to branch operations

Table 7.1 RISC processor specifications

CHAPTER 8

RTL DESIGN OF PROCESSOR

8.1 Introduction

The processor contains a number of basic blocks. There is a register bank of thirty two 32-biut registers, an ALU, a Shifter, a program counter, an Instruction register, a comparator, an address register and instruction decoder. All these units communicate through common 32-bit tristate data bus. A block diagram is shown in Figure 8.1

8.2 Arithmetic Logic Unit

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 is shown in Figure 8.2

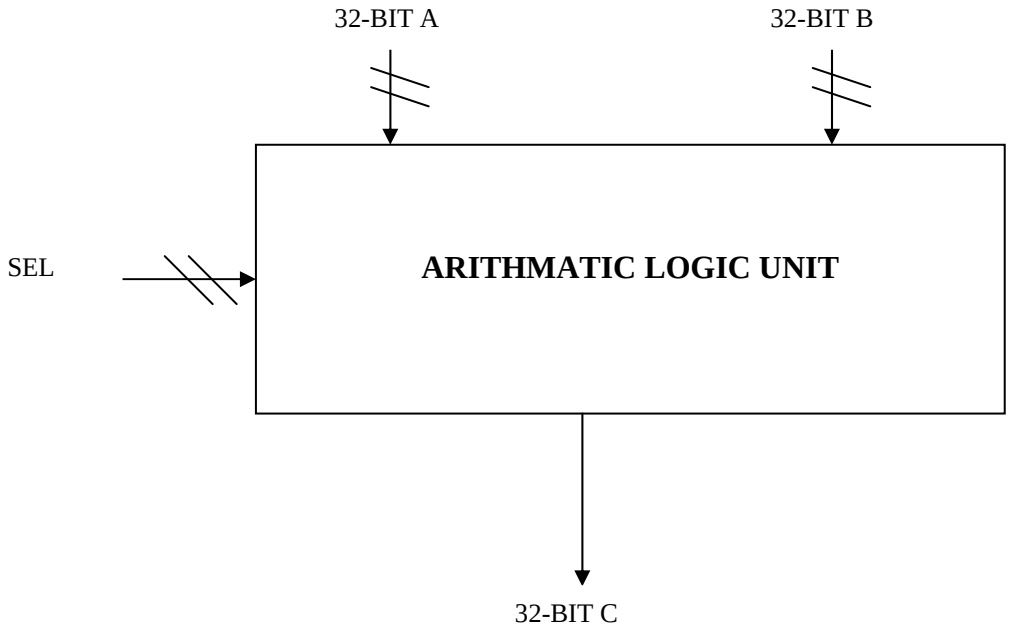

Figure 8.2 ALU

Inputs A and B are the two 32-bit input busses upon which the ALU Operations are performed. Output bus C returns the result of the ALU operation. Input SEL determines which operation is performed as specified by Table 8.1

Block diagram of processor

Sel Input	Operation	Description
0000	C<=A	Data pass
0001	C<=A AND B	Addition
0010	C<=A OR B	Subtraction
0011	C<=NOT A	Logical NOT
0100	C<=A XOR B	Logical AND
0101	C<=A + B	Logical OR
0110	C<=A - B	Logical XOR
0111	C<=A +1	Increment
1000	C<=A -1	Decrement
1001	C<=0	Zero

Table 8.1 ALU Function Table

As we can see, the ALU can perform some arithmetic operations such as add & subtract, and some logical operations such as AND OR, XOR and NOT

8.3 Comparator

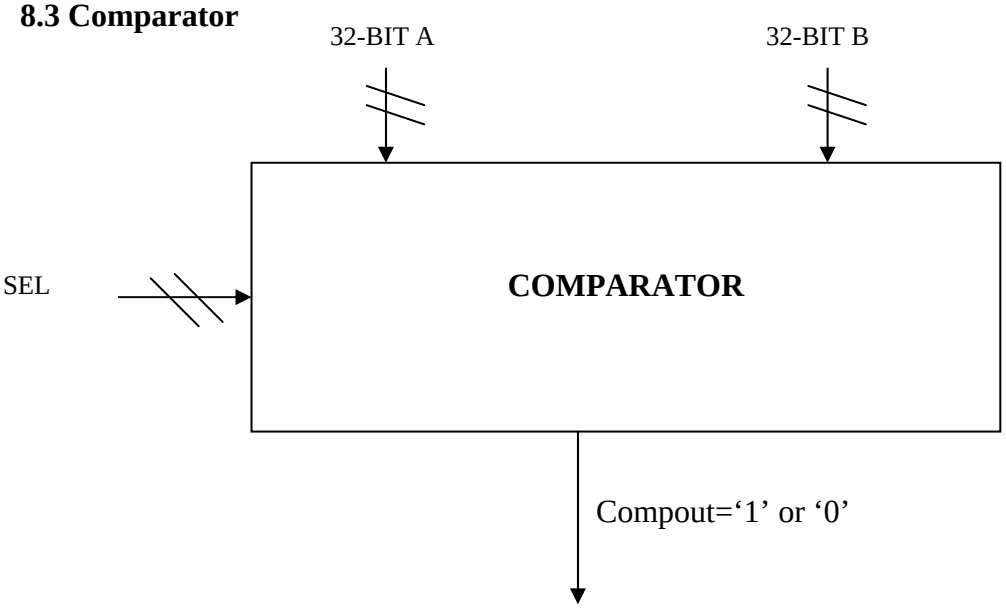

Figure 8.3 Comparator

The next component described is the comparator. This entity compares two values and returns either a ‘1’ or ‘0’ depending on the type of comparison requested and values being compared. A symbol showing the ports of the comparator is shown in figure 8.3.
The comparison type is determined by the value on input port SEL. The full table of comparison is shown in Table 8.2

SEL input	Comparison
EQ	Compout=’1’ when A equals B
NEQ	Compout=’1’ when A is not equal B
GT	Compout=’1’ when A is greater than B
GTE	Compout=’1’ when A is greater than or equal to B
LT	Compout=’1’ when A is less than B
LTE	Compout=’1’ when A is less than or equal to B

Table 8.2-comparison operation table

8.4 Register

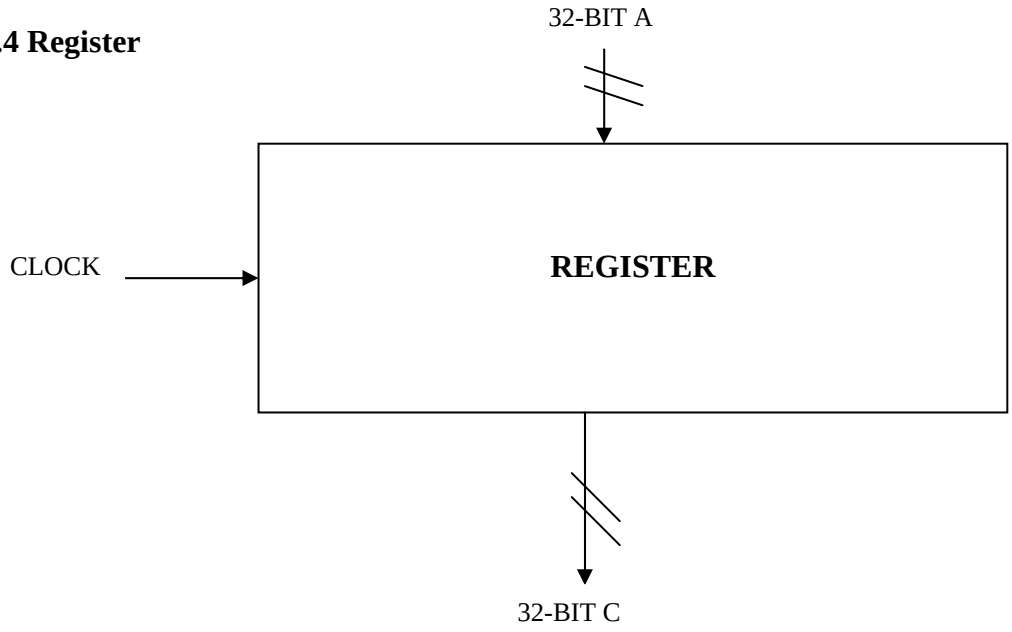

Figure 8.4 Register

The register entity is used for address register and instruction register. These registers need to be able to capture the input data (32-bit A) on a rising edge of the clock input and

drive output C with the captured data. Thus the value of input A is assigned to output C when a rising edge occurs on input clock. A symbol for the register entity is shown in figure 8.4

8.5 Tri-State Register

This is one more component of the CPU. The tristate register is connected to main data bus and can store information from the data bus as well as drive information to the data bus. It has four ports as shown in Figure 8.5

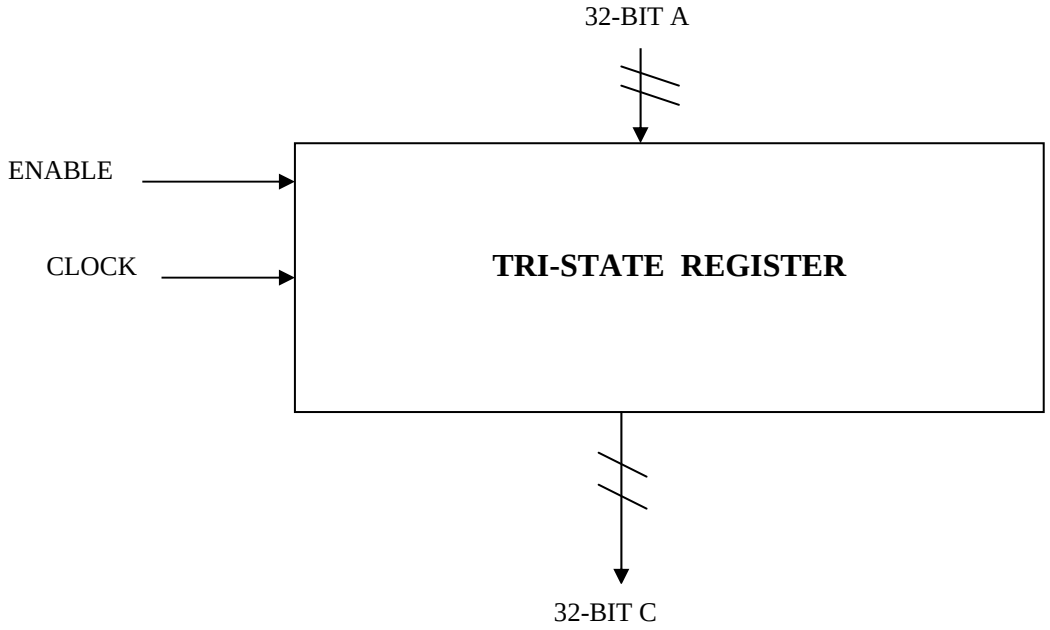

Figure 8.5 Tristate Register

Input A is data input to the register, and port C is the data output from register. Input clock is used to store a new value into register. When a rising edge is applied to input clock, the data on input A is stored in the register. The Enable is used to control output C. when enable is a '1' value the register state is driven to output C . when it is a '0', output C is a high impedance value and not driving.

8.6 Register Array

There are 32# of 32-bit registers. The register array 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 thirty-two 32-bit words. The symbol for the register array is shown in figure 8.6. To write a location in the regarray, set the input SEL to the location to be written, input data with the data to be written, and put the rising edge on input clock. To read a location from regarray, set the input SEL to the location to be read and set input enable to a '1'. The data is output on port C

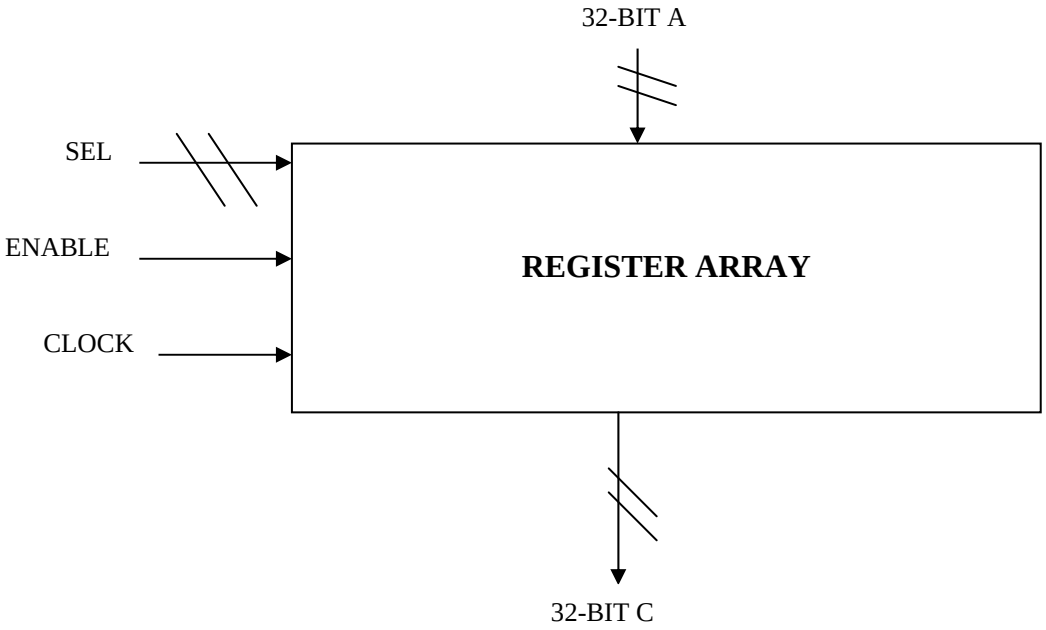

Figure 8.6 Register Array

8.7 Shift And Rotate Logic Circuit

The next device to be described is the shift rotate entity. This entity is used to perform shifting and rotation operations within the CPU. Te shift rotate entity has a 32-bit input bus(A) , a 32-bit output bus(C) and SEL input that determines which shift/rotate operation to perform. This entity is shown by the symbol in Figure 8.7

Shift pass mode allows the shifter to pass the input data to the output without any shift operations. This mode is quite common because all of the ALU operations flow through the shift entity. Table 8.3 shows a list of operations, which can perform by shift rotate logic.

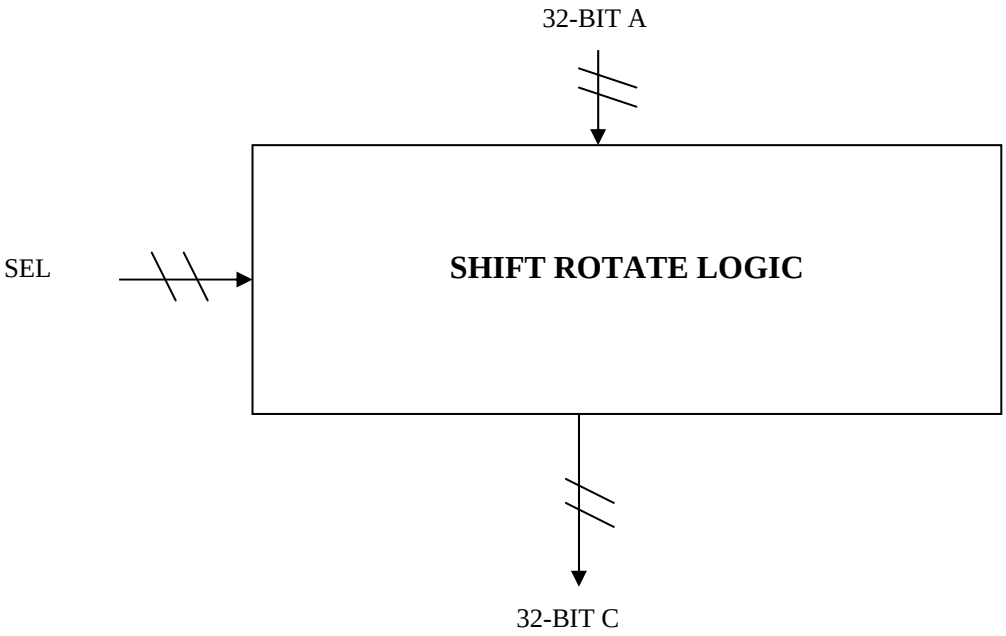

Figure 8.7 Shift Rotate Entity

SEL	Operation
Shiftpass	Shift pass
SHL	Shift left
SAR	Shift right (logical)
SAR	Shift right (arithmetic)
ROTL	Rotate Left
ROTR	Rotate right

Table 8.3-shift operations

8.8 Instruction Decoder And control logic

Instruction Decoder entity provides the necessary signal interactions to data flow properly through CPU and performs the expected functions. This design is based on the state machine. For state machine the next state is produced based on the current state and input signals. The instruction decoder unit has very less no.s of inputs and large no.s of output. A symbol for the instruction decoder is shown in Figure 8.8

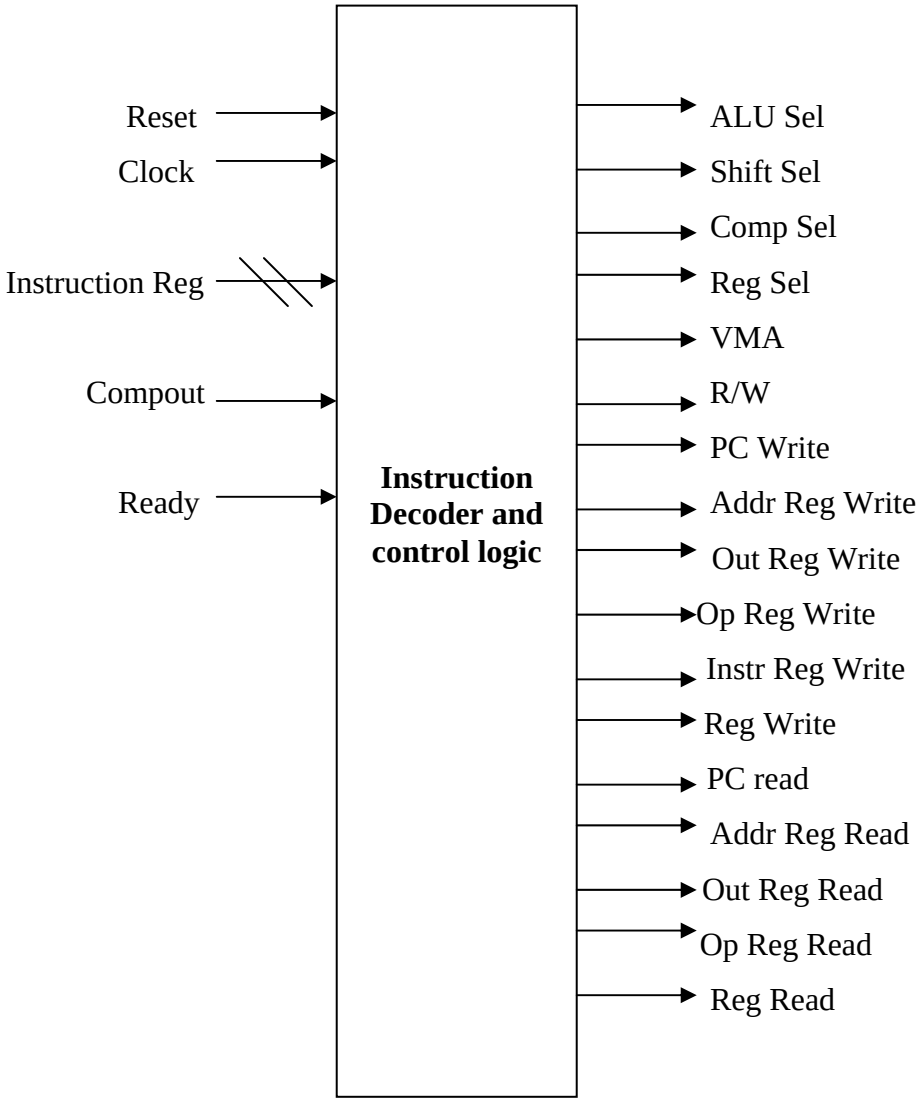

Figure 8.8 Instruction Decoder Entity

Instructions can be divided into a number of different types as follows:

Load-these instructions load register values from other registers, memory locations, or with immediate value given in the instruction.

Store- these instructions store register values to memory locations.

Branch-these instructions cause the processor to go to another location in the instruction stream. Some branch instructions test values before branching; others branch without testing.

ALU- these instructions perform arithmetic and logical operations such as ADD, SUBTRCT, OR, AND, and NOT.

Shift-These instructions use the shift unit to perform shift operations on the data passed to it.

8.8.1 Types of instructions:

1. Data transfer instructions
2. Arithmetic instructions
3. Logical instruction
4. Shift-rotate instructions
5. Branch instructions

8.8.2 Instruction Set

Following Table 8. 4 shows the details of instructions

Types	Instruction		Description	Opcode (D31-D27)
Data transfer instructions	NOP		No operation	00000
	LOAD	R from memory	Load register (memory address is specified by register)	00001
	STORE	R to memory	Store register (memory address is specified by register)	00010
	MOVE	R to R	Move register to register	00011

	LODI	Data to R	Load register with immediate value	00100
Arithmetic instructions	ZERO	R	Zero a register	01111
	ADD	R1 R2 R3	R1+R2->R3	01101
	SUB	R1 R2 R3	R1-R2->R3	01110
	INC	R	Increment	00111
	DEC	R	Decrement	01000
Logical instruction	NOT	R	Not a register value	01100
	AND	R1 R2 R3	R1 and R2->R3	01001
	OR	R1 R2 R3	R1 or R2->R3	01010
	XOR	R1 R2 R3	R1 xor R2->R3	01011
Shift-rotate instructions	SHL	R1 R2	Shift left R1 -> R2	11010
	SHR	R1 R2	Shift right R1 -> R2	11011
	ROTL	R1 R2	Rotate left R1 -> R2	11101
	ROTR	R1 R2	Rotate right R1 -> R2	11100
Branch instructions	BRANCHI	ADDRESS	Direct unconditional branch	00101
	BRANCH	R	Indirect unconditional branch (address is in register)	10101
	BRANCHGTI	R1 R2 ADDRESS	Direct conditional branch If R1 is greater then R2	00110
	BRANCHGT	R3 R1 R2	Indirect conditional branch If R1 is greater then R2, (address is in register R3)	10100
	BRANCHGTEI	R1 R2 ADDRESS	Direct conditional branch If R1 is greater then or equal to R2	11110

	BRANCHGTE	R3 R1 R2	indirect conditional branch If R1 is greater then or equal to R2, (address is in register R3)	11111
	BRANCHLTI	R1 R2 ADDRESS	Direct conditional branch If R1 is less then R2	10000
	BRANCHLT	R3 R1 R2	Indirect conditional branch If R1 is less then R2, (address is in register R3)	10001
	BRANCHLTEI	R1 R2 ADDRESS	Direct conditional branch If R1 is less then or equal to R2	11000
	BRANCHLTE	R3 R1 R2	Indirect conditional branch If R1 is less then or equal to R2, (address is in register R3)	11001
	BRANCHEQI	R1 R2 ADDRESS	Direct conditional branch If R1 equal to R2	10111
	BRANCHEQ	R3 R1 R2	Indirect conditional branch If R1 is equal to R2, (address is in register R3)	10110
	BRANCHNEQI	R1 R2 ADDRESS	Direct conditional branch If R1 is not equal to R2	10011
	BRANCHNEQ	R3 R1 R2	Indirect conditional branch If R1 is not equal to R2, (address is in register R3)	10010

Table8. 4 Instruction Set

8.8.3 Instruction representation

All instructions contain the opcode in five most significant bits of the instruction. Single word instructions also contain one or two or three 5-bit register field at LSB side

e.g.

MOV R1, R2 (1-word instruction)

5-BIT OPCODE (5-bit)	XXXXX……XX	CODE FOR R1(5-bit)	CODE FOR R2(5-bit)

ADD R1, R2, R3 (1-word instruction)

5-BIT OPCODE (5-bit)	XXXXXX ……XX	CODE FOR R1(5-bit)	CODE FOR R2(5-bit)	CODE FOR R2(5-bit

BRANCHI ADRDRESS (2-word instruction)

5-BIT OPCODE (5-bit)	XXXXXX……XX
ADRDRESS (32-bit)	

8.9 System Level Design

If the instruction is an add of two registers, the decoder would cause the first value to be written to Opreg for temporary storage. The second register value would be placed on the data bus. The ALU would be placed in add mode and result would be stored in register OutReg. Outreg would store the resulting value until it is copied to the final destination.

When executing an instruction, a number of steps take place. The PC holds the address in memory of current instruction. After an instruction has finished execution, the PC is advanced to where the next instruction is located. If the processor is executing a linear stream of instructions, this is the next instruction. If a branch was taken, the PC is loaded with the next instruction location directly

Instruction decoder copies the PC value to the address register, which outputs the new address on the address bus. At the same time decoder sets R/W (read write signal) to a '0' value for a read operation and sets VMA to a '1', signaling the memory that the address is now valid. The memory decodes the address and places the memory data on the data bus , when the data has been placed on the data bus, the memory has set the ready signal to a '1' value indicating that the memory data is ready for consumption.

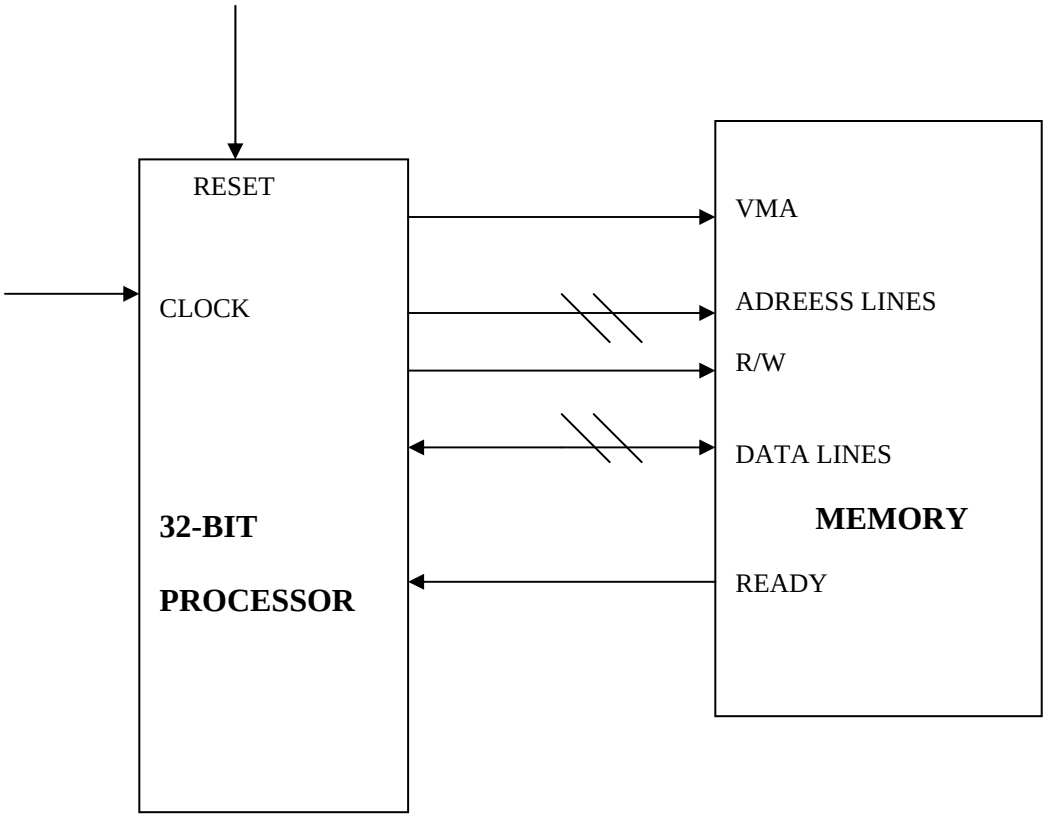

Figure 8.9-system level Design

The decoder causes the memory data to be written into instruction register. The decoder now has access to the instruction and decodes the instruction. The decoded instruction executes, and the process starts over again

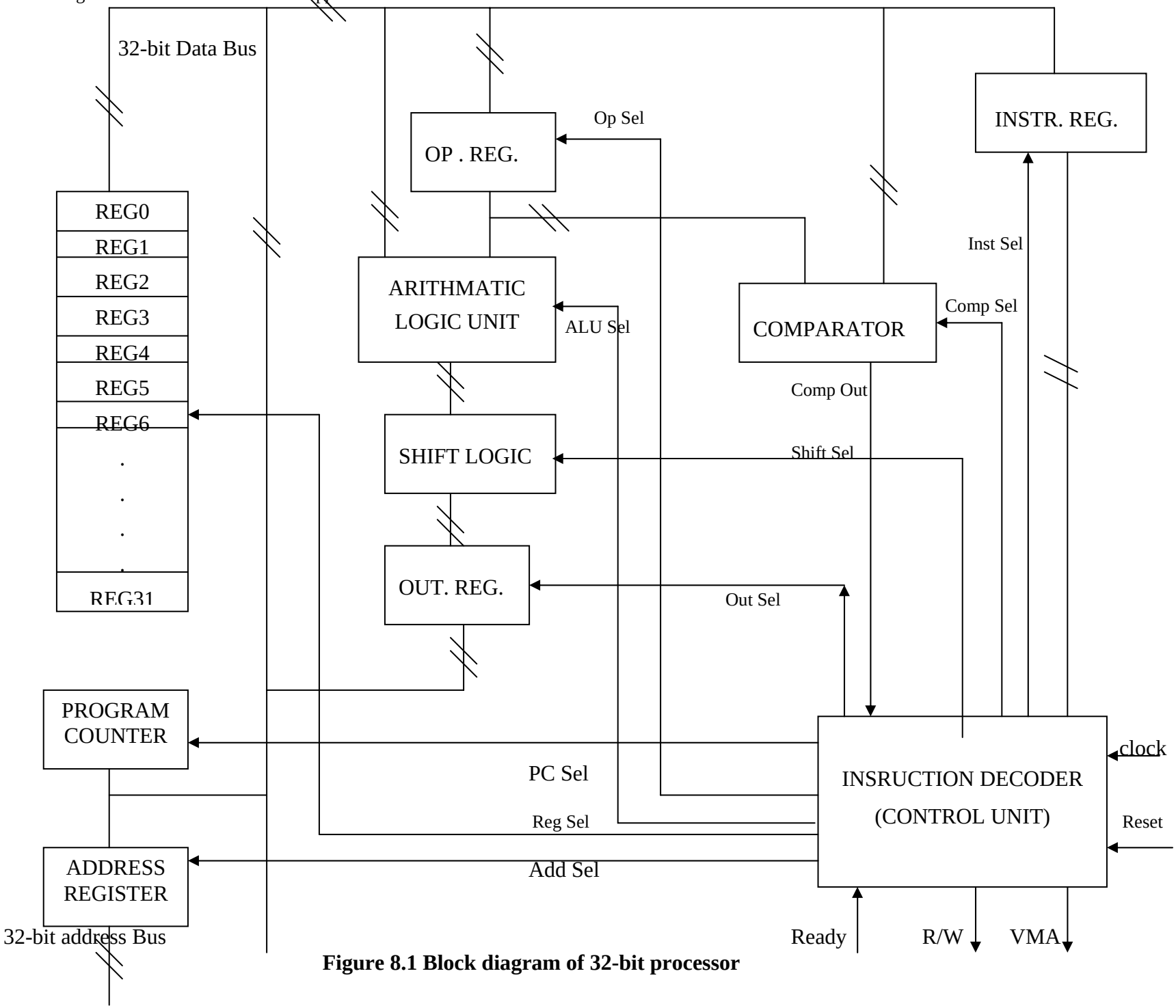

Figure 8.1 Block diagram of 32-bit processor

CHAPTER 9

RESULTS AND ANALYSIS

9.1 Synthesis result

Device utilization summary:

Selected Device: v300pq240-6

Number of Slices:	2597	out of	3072	84%
Number of Slice Flip Flops:	2984	out of	6144	48%
Number of 4 input LUTs:	3787	out of	6144	61%
Number of bonded IOBs:	2	out of	170	1%
Number of TBUFs:	161	out of	3072	5%
Number of GCLKs:	1	out of	4	25%

Total memory usage is 132580 kilobytes

9.2 block of data transfer

Problem statement for system

The program (shown in Appendix-b) will copy the block of data available at memory location 0000008H to 0000001B to new memory locations starting from 00000020H and onwards

Simulation results showing the address and data, VMA. Ready and R/w signals

9.2.1 Initial phase

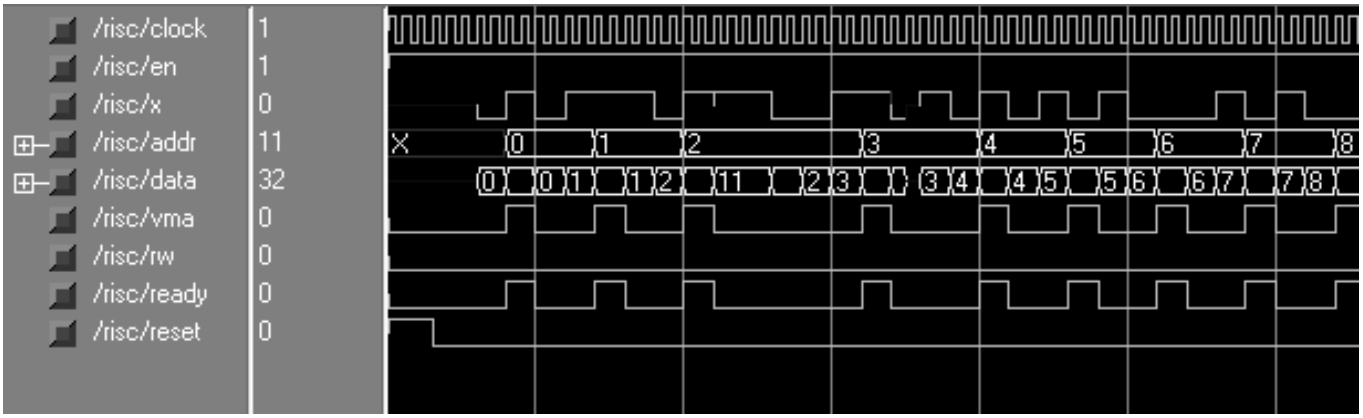

Figure 9.1 simulation result for data transfer program (initial phase)

9.2.2 Phase shows the repetition of execution

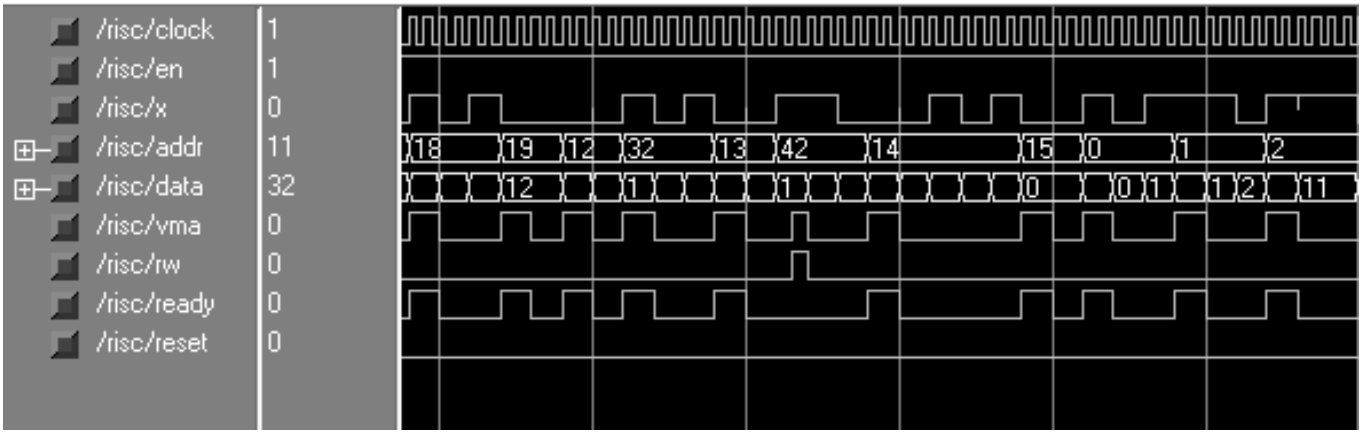

Figure 9.2 simulation results for data transfer program (phase shows repetition)

Various control signals generated by Instruction decoder

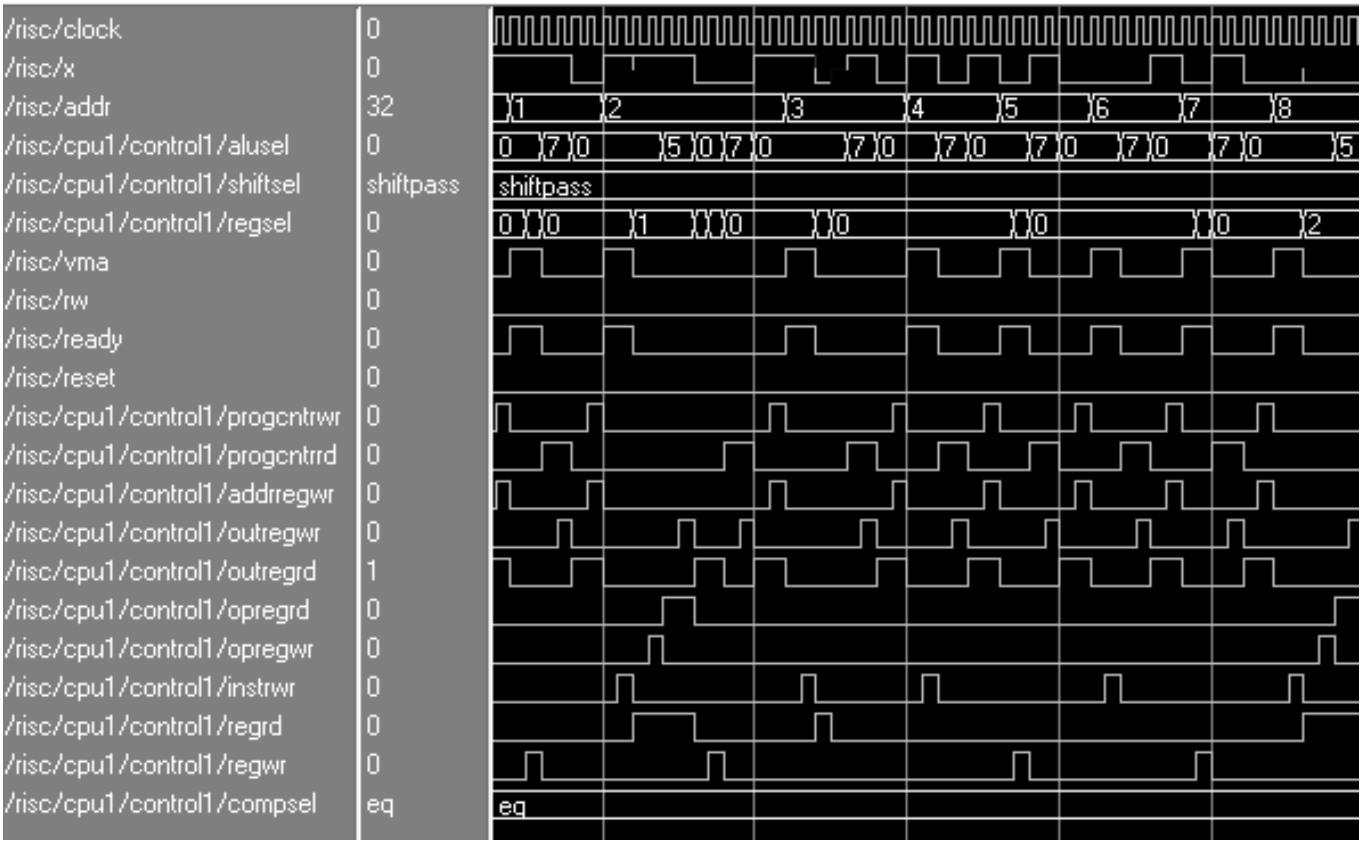

Figure 9.3 simulation for data transfer program (control signals from instruction decoder)

Contents of memory before data transfer

// Memory data file (do not edit the following line - required for mem load use)
// Instance=/processor/mem1/mem_data
// Format=mti address radix=h data radix=s version=1.0

0: 00010000000000000000000000000001 00000000000000000000000000001011
2: 00110100000000000000010000100001 00101100000000000000010000100101
4: 00010000000000000000000000000011 00000000000000000000000000001010
6: 00010000000000000000000000000010 00000000000000000000000000010000
8: 00110100000000000000100001000010 00010000000000000000000000000110
a: 00000000000000000000000000100000 00100000000000000000000000000110
c: 00000100000000000000000000100100 00001000000000000000000010000010
e: 01010000000000000001010000100110 00011100000000000000000000000001
10: 00011100000000000000000000000010 00010100000000000000000000001111
12: 00000000000000000000000000001100 00000000000000000000000000000000
14: 00000000000000000000000000000000 00000000000000000000000000000001
16: 00000000000000000000000000000010 00000000000000000000000000000100
18: 00000000000000000000000000001000 00000000000000000000000000010000
1a: 00000000000000000000000000100000 00000000000000000000000001000000
1c: 00000000000000000000000010000000 00000000000000000000000100000000
1e: 00000000000000000000001000000000 00000000000000000000010000000000
20: 00000000000000000000000000000000
00000000000000000000000000000000
22: 00000000000000000000000000000000
00000000000000000000000000000000
24: 00000000000000000000000000000000
00000000000000000000000000000000

26: 00000000000000000000000000000000
00000000000000000000000000000000
28: 00000000000000000000000000000000
00000000000000000000000000000000
2a: 00000000000000000000000000000000
00000000000000000000000000000000
2c: 00000000000000000000000000000000 00000000000000000000000000000000
2e: 00000000000000000000000000000000 00000000000000000000000000000000
30: 00000000000000000000000000000000 00000000000000000000000000000000
32: 00000000000000000000000000000000 00000000000000000000000000000000

Contents of memory before data transfer

// Memory data file (do not edit the following line - required for mem load use)
// Instance=/processor/mem1/mem_data
// Format=mti address radix=h data radix=s version=1.0

0: 00010000000000000000000000000001 00000000000000000000000000001011
2: 00110100000000000000010000100001 00101100000000000000010000100101
4: 00010000000000000000000000000011 00000000000000000000000000001010
6: 00010000000000000000000000000010 00000000000000000000000000010000
8: 00110100000000000000100001000010 00010000000000000000000000000110
a: 00000000000000000000000000100000 00100000000000000000000000000110
c: 00000100000000000000000000100100 00001000000000000000000010000010
e: 01010000000000000001010000100110 00011100000000000000000000000001
10: 00011100000000000000000000000010 00010100000000000000000000001111
12: 00000000000000000000000000001100 00000000000000000000000000000000
14: 00000000000000000000000000000000 00000000000000000000000000000001
16: 00000000000000000000000000000010 00000000000000000000000000000100
18: 00000000000000000000000000001000 00000000000000000000000000010000

1a: 00000000000000000000000100000 00000000000000000000001000000

1c: 00000000000000000000010000000 00000000000000000000100000000

1e: 00000000000000000001000000000 00000000000000000010000000000

20: 00000000000000000000000000000010

00000000000000000000000000000100

22: 00000000000000000000000000001000

00000000000000000000000000010000

24: 00000000000000000000000000100000

00000000000000000000000001000000

26: 00000000000000000000000010000000

00000000000000000000000100000000

28: 00000000000000000000001000000000

00000000000000000000010000000000

2a: 00000000000000000000000000000010

00000000000000000000000000000000

2c: 00000000000000000000000000000000 00000000000000000000000000000000

2e: 00000000000000000000000000000000 00000000000000000000000000000000

30: 00000000000000000000000000000000 00000000000000000000000000000000

32: 00000000000000000000000000000000 00000000000000000000000000000000

9.3 Shift logic

Simulation result

Figure 9. 4 Simulation result for shifter

Chip scope result

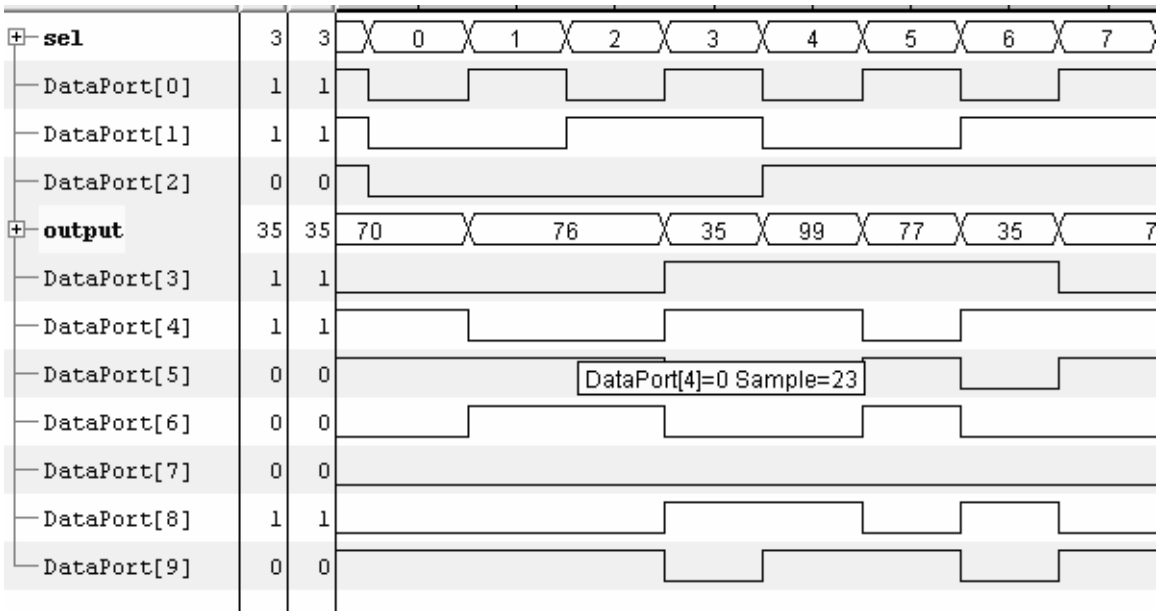

Figure 9. 5 Chip scope result for shifter

9.4 ALU Result

Simulation result

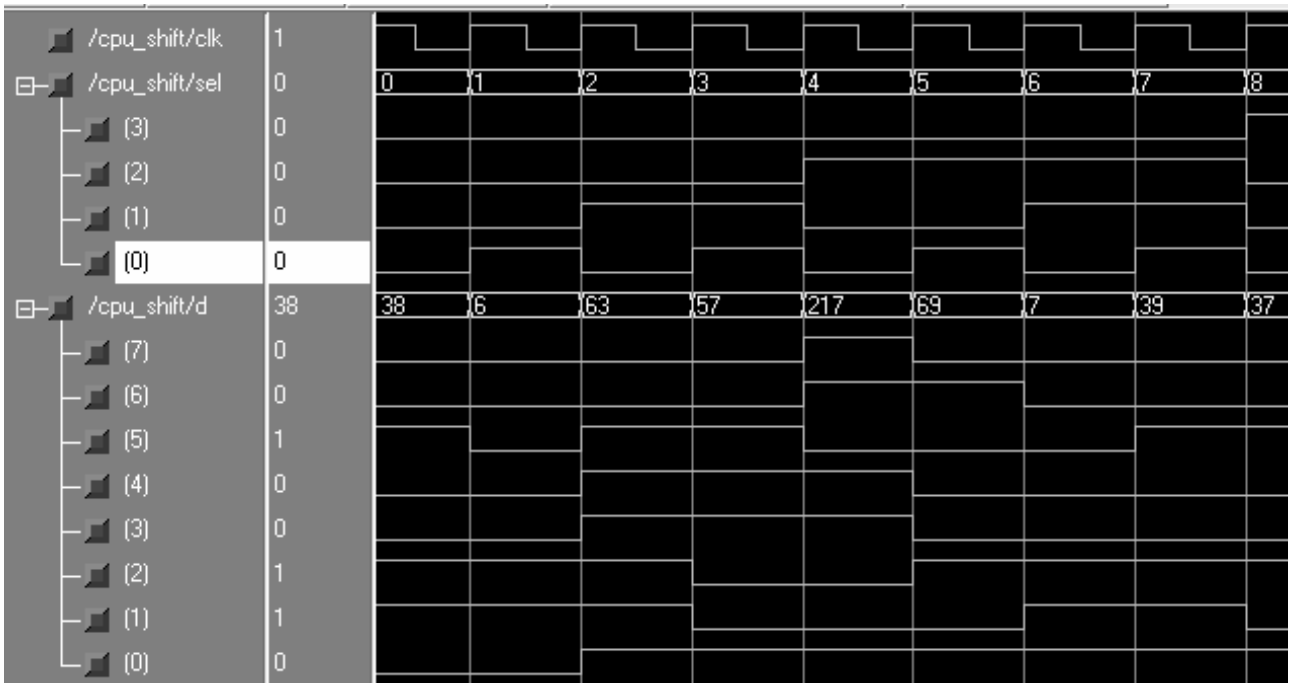

Figure 9. 6 simulation result for ALU

Chip scope result

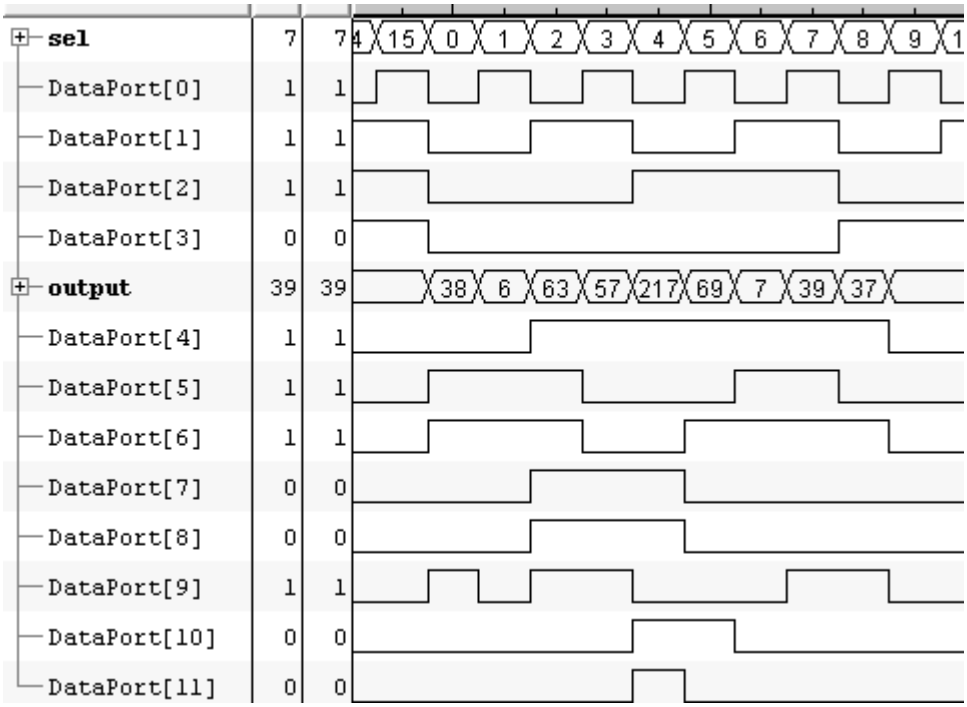

Figure 9. 7 Chip scope result for ALU

CHAPTER 10

SUMMARY AND FUTURE SCOPE OF WORK

10.1 Summary

The project work entitled as “RTL design of RISC processor for DSP Application” is selected to carry out the work during September 2006-April 2007. During first phase of project work, literature survey is carried out to understand the basic architecture of Processor, general block diagram of the same, function of each block, instruction set of processor. The detailed study of various processors like ARM, NIOS_II, and PowerPC is also carried out

RTL coding for RISC processor has been carried using VHDL. The design was download on Xilinx FPGA. The functionality is verified by executing a program for processor to transfer a block of data from memory to memory through register of register bank. The individual blocks like ALU and Shifter are simulated using Modelsim simulator. After down loading the same on the FPGA, the result is verified using Chip scope.

10.2 Future work

As in this dissertation work, the RTL design of RISC processor has been carried out; there is scope of further work. It is possible to design the DSP module and integration of the same can be done with designed processor. As memory is designed on same FPGA on which RISC processor is designed, it is also possible to enhance the performance of RISC processor by allocating more are of programmable Logic device and interface the same with external memory. The work can be extended by value addition in form to support the exceptions, enhancing the instruction set also.

REFERENCES

1. Modern Processor Design by John Paul Shen, Mikko H. Lipasti
2. Computer Architecture A Quantitive Approach by John Hennesy and D Patterson
3. Advanced digital Design with the Verilog HDL by Michael D. Ciletti.
4. Nios_II Processor Reference Handbook
5. Nios_II Custom instruction user guide
6. Arm system developer's guide by Sloss, Chris Wright
7. Arm Architecture reference manual edited by David Seal
8. Arm system-on-chip architecture by Steve Furber
9. PowerPC processor user guide
10. http://www.xilinx.com
11. http://www.altera.com
12. http://www.arm.com

APPENDIX-A

RISC VS CISC

A. What is RISC?

RISC (Reduced Instruction Set Computer): A computer architecture that reduces chip complexity by using simpler instructions. RISC compilers have to generate software routines to perform complex instructions that were previously done in hardware by CISC computers. In RISC, the microcode layer and associated overhead is eliminated.

RISC keeps instruction size constant, bans the indirect addressing mode and retains only those instructions that can be overlapped and made to execute in one machine cycle or less. The RISC chip is faster than its CISC counterpart and is designed and built more economically.

B. What is the comparison between RISC and CISC?

The main characteristics of CISC microprocessors are:

1) Extensive instructions.
2) Complex and efficient machine instructions.
3) Micro encoding of the machine instructions.
4) Extensive addressing capabilities for memory operations.
5) Relatively few registers.

RISC processors have following traits:

1) Reduction of the instruction set.
2) Instruction pipelining (the interleaved execution of many instructions).
3) Load/store architecture (only the load and store instructions have access to memory, all others work with the internal processor registers).
4) Unity of RISC processors and compilers (the compiler is no longer developed for

a specific chip, but instead, at the outset, the compiler is developed in conjunction with the chip to produce one unit).

C. Explain the performance evaluation of RISC and CISC

The following equation is commonly used for expressing a computer's performance ability:

TIME / PROGRAM = (TIME / CYCLE) * (CYCLES / INSTRUCTION) * (INSTRUCTIONS / PROGRAM)

The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program.

RISC Vs CISC instruction execution flow

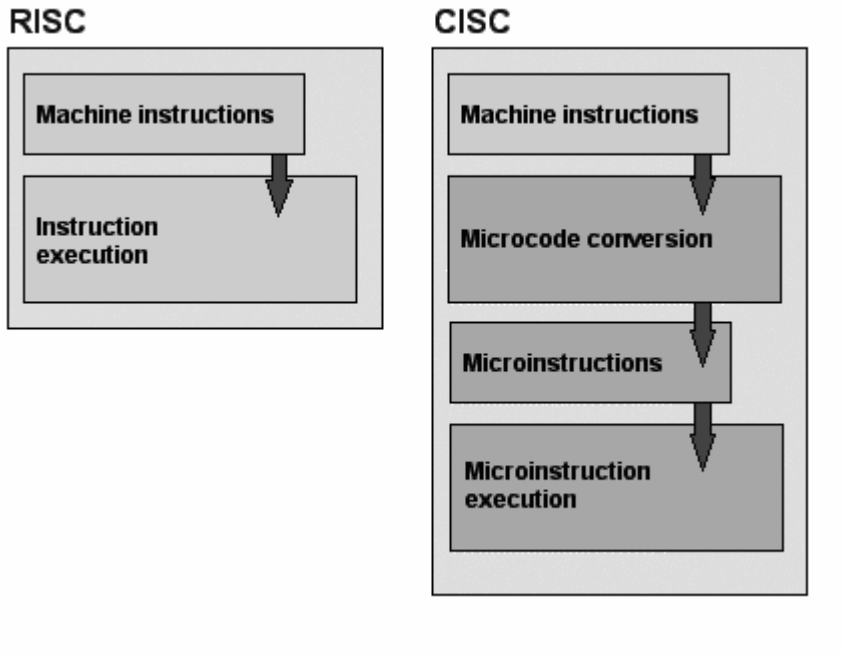

Figure A.1 RISC Vs CISC instruction execution flow

Block Diagram of typical RISC Processor

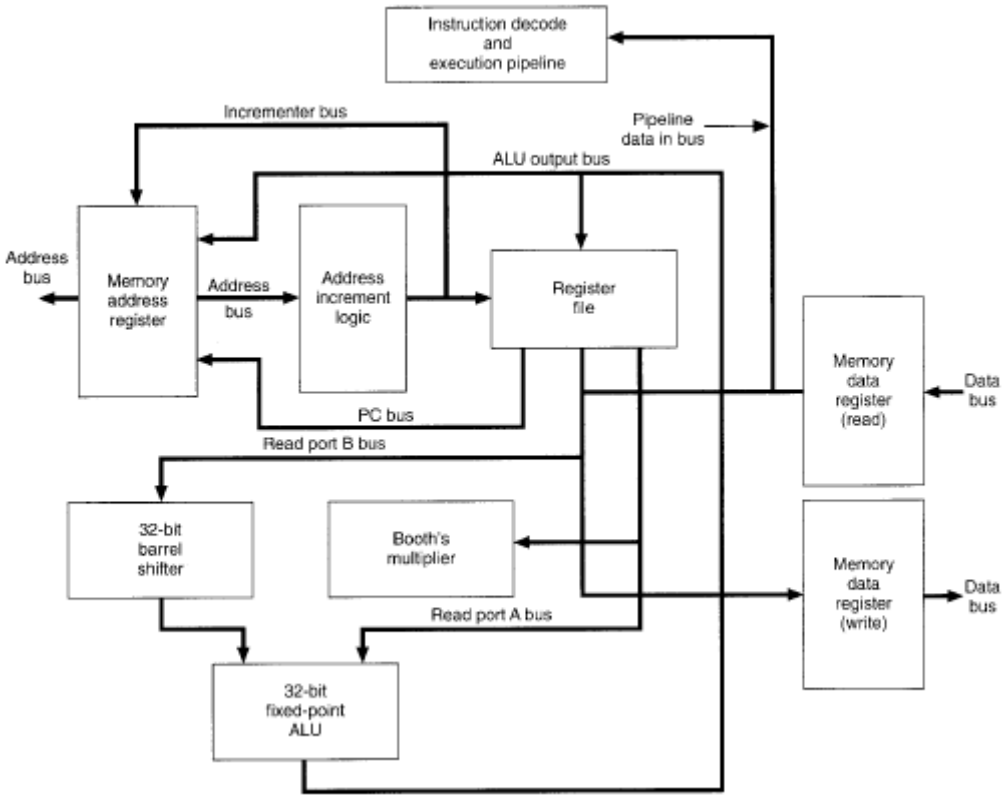

Figure A.2 Block Diagram of typical RISC Processor

APPENDIX-B

Assembly language Program to transfer a block of data

This program will copy the block of data available at memory location 0000008H to 0000001B to new memory locations starting from 00000020H and onwards

label	Instruction	description
XX:	LOADI R1,08H	Starting address of block of data is stored is R1
	ADD R1,R1,R1	R1+R1->R1 (New Content Will Be 10H)
	LOADI R5,00H	Starting address of program is stored in R5
	LOADI R2,10H	Starting address of memory where data is to be transferred is stored is R2
	SHL R2, R2	Shift the content of R2 by 1 (new content in R2 is 20H)
	LOADI R6,1BH	Last address of block of data is stored is R6
	DEC R6	Decrement R6 by 1 (new content in R6 is 1AH)
YY:	LOAD R4, R1	Load the data in dummy register R4 from memory location whose address is in R1
	STORE R2 ,R4	Store the data from dummy register R4 to memory location whose address is in R2
	BRANCHGTI R1,R6, R5	Indirect jump to memory location specified by R5 based on content of R1 and R6
	INC R1	To locate next memory location from where new data is to be transferred
	INCR2	To locate next memory location to where new data is to be transferred
	BRANCHI YY	To jump for next data transfer

Table A.1 Assembly Language programme