

IMPLEMENTATION OF DSP PROCESSOR ON FPGA

A DISSERTATION

*Submitted in partial fulfillment of the
requirements for the award of the degree*

of

MASTER OF TECHNOLOGY

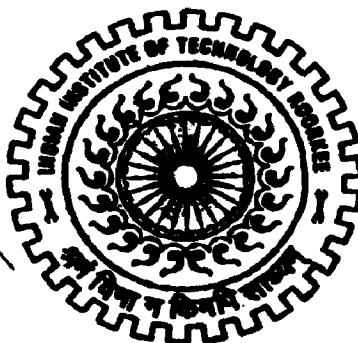
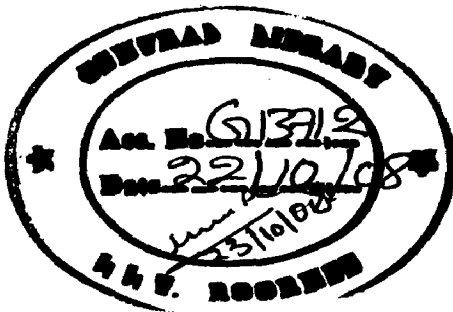
in

ELECTRICAL ENGINEERING

(With Specialization in System Engineering and Operations Research)

By

T.RAVI KUMAR



18

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667 (INDIA)

JUNE, 2008

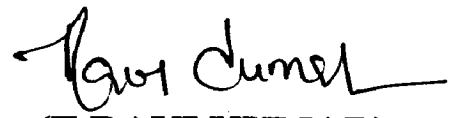
CANDIDATE'S DECLARATION

I hereby declare that the work which is being presented in this dissertation entitled, "IMPLEMENTATION OF DSP PROCESSOR ON FPGA" submitted in the partial fulfillment of the requirements for the award of the degree "Master of Technology" with specialization in System Engineering and Operations Research, to the Department of Electrical Engineering, IIT Roorkee, Roorkee is an authentic record of my own work carried out during the period from August 2007 to June 2008 under the supervision of Dr. Indra Gupta, Associate Professor, Department of Electrical Engineering, IIT Roorkee, Roorkee.

The matters embodied in this report have not been submitted by me for the award of any other degree or diploma.

Date: 30 June 2008

Place: Roorkee


(T.RAVI KUMAR)

This is to certify that above statement made by candidates is correct to the best of my knowledge.

Date: 30 June 2008

Place: Roorkee


(Dr. Indra Gupta)

Associate Professor,
Electrical Engineering Department,
Indian Institute of Technology
Roorkee.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my supervisor, **Dr. Indra Gupta**, Associate Professor, Department of Electrical Engineering, for the patience and guidance throughout the entire duration of my thesis. Continuous monitoring and time management was an inspiring force for me to complete the work. Without her supervision, this thesis would never have been a success. Working under her guidance has been a great experience which has given me a deep insight in the area of technical research. Her painstaking support and involvement in preparation of manuscript, theoretical analysis and simulation studies are gratefully acknowledged. I humbly acknowledge a lifetime's gratitude to her and hope for a continued interaction even in the future.

I consider myself extremely lucky and privileged to learn about Micro Processors subject from **Prof. M. K. Vasantha**. It is because of him, that I started developing interest in Digital electronics and Computer architecture. His way of teaching, presentation in the classroom, the discipline he has inculcated in us, will be what I would like to follow throughout my life. His constant encouragement and willingness to listen to and help with my academic queries are some of the things I benefited with.

I also thank **Dr. H.O Gupta**, **Dr. Surendra Kumar**, **Dr. Rajendra Prasad**, **Dr. G.N. Pillai** and **Dr. Barjeev Tyagi** for extending moral support and technical discussions as and when required during the work.

I would like to take this opportunity to express my deep sense of gratitude to my family for their support and encouragement they have provided me over the years.

I would also like to thank my friends and lab assistants who have offered me their unrelenting assistance throughout the course.

Date: 30 June 2008

T.Ravi Kumar

Place: Roorkee.

Abstract

Microprocessors built specifically for digital signal processing are DSP processors. DSP chips are high-speed, dedicated microprocessors that have been optimized to perform Arithmetic operations on the huge amounts of data required by spectral analysis and signal processing algorithms.

DSP Processors are the computational engines on which DSP applications are built. General purpose DSP processors provides an effective way to design and implement DSP algorithms for real-world applications. DSP is one of the core technologies in rapidly growing applications like communications and audio processing. DSP systems have progressed remarkably in the past decades, especially within the last few years. The estimated growth of DSP processors in the last 6 years is over 40%. The variety of DSP capable processors for various applications also increased with the rising popularity of DSP processors.

The present work is a structured approach to design and implementation of an embedded DSP processor core. The work focuses on the design of the 16bit DSP Processor described by the VHDL language and optimized by forcing timing constraints. The proposed DSP Processor is implemented and tested through simulation results and the results of the processor are validated.

The designs are coded with VHDL, synthesized and configured into Spartan-2 XC2S200-5Q208 FPGA, from Xilinx family. The ISE 7.1 synthesis tool is used in this project for synthesis and implementation.

CONTENTS

	Page no
CANDIDATE DECLARATION	I
ACKNOWLEDGEMENT	II
ABSTARCT	III
CONTENTS	IV
LIST OF FIGURES	VI
CHAPTER 1: Introduction	1
1.1 General Introduction	1
1.1.1 A Brief History of FPGAs	3
1.1.2 What can FPGAs be used for	4
1.2 Objectives of the dissertation	5
1.3 Organization of Thesis	5
CHAPTER 2: DSP Processors	7
2.1 Introduction to DSP Processors	7
2.1.1 DSP Environment	7
2.1.2 Applications	8
2.1.3 DSP Architecture	9
2.2. General Architecture of DSP Processors	10
2.2.1 Data Path	11
2.2.2 Control unit	12
2.2.3 Memory	12
CHAPTER 3: Developed Processor: Instruction Set Architecture and Addressing Modes	13
3.1 Introduction	13
3.2 Instruction Format	13
3.3 Addressing Modes	14
3.4 Instruction Types	15

3.4.1 R-type instructions	15
3.4.2 I-type instructions	15
3.4.3 J-type instructions	16
3.5 Instruction Set	17
3.5.1 Data transfer instructions	18
3.5.2 Arithmetic instructions	19
3.5.3 Logical instructions	19
3.5.4 T register, P register and Multiply instructions	20
3.5.5 Shift instructions	20
3.5.6 Branching instructions	21
CHAPTER 4: Design Architecture of Proposed DSP Processor	22
4.1 Introduction	22
4.2 Architecture Overview	22
4.3 Data Path (DP)	25
4.3.1 Arithmetic and Logic Unit (ALU)	25
4.3.2 Comparator	26
4.3.3 Multiplier	27
4.3.3.1 Array-Based Multiplication	30
4.3.4 Register File	33
4.3.5 Multiplexer	34
4.3.6 Shifters	35
4.3.7 Reg	37
4.3.8 Trireg	37
4.4 Controller	38
4.5 CPU	41
CHAPTER 5: Simulation Results	43
CHAPTER 6: Conclusion and Future Scope	50
REFERENCES	52
APPENDIX A	54
APPENDIX B	59
APPENDIX C	62

List of Figures

	Page No.
Figure 2.1. DSP System Overview	8
Figure 2.2. Von Neumann Architecture	9
Figure 2.3. Harvard Architecture	9
Figure 2.4 General Architecture of DSP Processor	10
Figure 3.1. Instruction format	13
Figure 3.2 Fields in the R-type instruction	15
Figure 3.3 Data path for an R-type instruction	15
Figure 3.4 Fields in an I-type instructions	16
Figure 3.5. Data path for an I-type instruction	16
Figure 3.6 Instruction fields for Jump instruction	16
Figure 3.7 Data path for Jump instruction	17
Figure 4.1. Internal Architecture of designed DSP Processor	23
Figure 4.2 ALU	25
Figure 4.3. Symbol of Comparator	26
Figure 4.4. Architecture of comparator	27
Figure 4.5. Multiplier Block diagram	28
Figure 4.6 A 16-bit Array Multiplication	29
Figure 4.7. Full Adder symbol	30
Figure 4.8. Full adder RTL diagram	30
Figure 4.9. A16 Bit Array Multiplier	31
Figure 4.10. Multiplier Internal Architecture	32

Figure 4.11. Register File Symbol	33
Figure 4.12. Register File internal architecture	33
Figure 4.13. Block diagram representation of 2-to-1 MUX	34
Figure 4.14. Multiplexer	34
Figure 4.15. Block diagram of Barrel Shifter	35
Figure 4.16 RTL Diagram of Barrel Shifter	36
Figure 4.17. Register Symbol and RTL diagram	37
Figure 4.18 Tri Register Symbol and RTL diagram	38
Figure 4.19. Block diagram of the Control Path	39
Figure 4.20. Controller Block diagram	40
Figure 4.21 RTL Diagram of CPU	41
Figure 5.1 Simulation Result of ALU	43
Figure 5.2 Simulation Result of B Shifter	44
Figure 5.3 Simulation Result of Tri Register	44
Figure 5.4 Simulation Result of Register Array	45
Figure 5.5 Simulation Result of Control Unit	45
Figure 5.6 Simulation Result of 16 bit Array Multiplier	46
Figure 5.7(a) Simulation report of DSP Processor for case 1	47
Figure 5.7 (b) Simulation report of DSP Processor for case 2	48
Figure 5.7 (c) Simulation report of DSP Processor for case 3	48
Figure 5.7 (d) Simulation report of DSP Processor for case 4	49

Chapter 1

Introduction

1.1 General Introduction

A signal is an impulse or a fluctuating electric quantity, such as voltage, current, or electric field strength, whose variations represent coded information. Electrical signals are normally either discrete in time and amplitude or continuous in time and amplitude. Digital signals are a sequence of quantized values making them discrete in time and amplitude. The digital signal [5,18] amplitude is represented by limited number of predefined values unlike continuous values where it is samples continuously. To convert the continuous signal into a digital signal an Analog to Digital Converter (ADC) can be used. Once the signals are processed it needs to be converted back into the continuous domain for which a Digital to Analog Converter (DAC) is used. The ADC works by sampling the signal at fixed points and the DAC works by interpolation.

DSP stands for Digital Signal Processing. In practice DSP mean processing of digital signals using electronic circuits or computers such as DSP Processors or Application Specific Integrated Circuit (ASIC). Processing signals mean applying algorithms based on arithmetic computations. Arithmetic could be any arithmetic operations like addition, subtraction, and so on. DSP is the processing of analog signals in the digital domain [18]. Real-world signals, such as voltages, pressures, seismic vibrations, visual images, sound waves and temperatures, are converted to their digital equivalents at discrete time intervals for processing by the CPU of a digital computer. DSP is the mathematics, algorithms and the techniques used to manipulate these signals after they have been converted into a digital form. The world of science and engineering is filled with signals: images from remote space probes, voltages generated by the heart and brain, radar and sonar echoes, seismic vibrations, and countless other applications. DSP is the science of using computers to understand these types of data. This includes a wide variety of goals: filtering, speech recognition, image enhancement, data compression, neural networks, and much more. DSP is one

of the most powerful technologies that will shape science and engineering in the twenty-first century.

DSP is useful in almost any application that requires the high-speed processing of a large amount of numerical data. The data can be anything from position and velocity information for a closed loop control system, to two-dimensional video images, to digitized audio and vibration signals.

The roots of DSP are in the 1960s and 1970s when digital computers first became available. Computers were expensive during this era, and DSP was limited to only a few critical applications. Pioneering efforts were made in four key areas: radar & sonar, where national security was at risk; oil exploration, where large amounts of money could be made; space exploration, where the data are irreplaceable; and medical imaging, where lives could be saved. The personal computer revolution of the 1980s and 1990s caused DSP to explode with new applications. Rather than being motivated by military and government needs, DSP was suddenly driven by the commercial marketplace. Anyone who thought they could make money in the rapidly expanding field was suddenly a DSP vendor. DSP reached the public in such products as: mobile telephones, compact disc players, and electronic voice mail.

DSP Processors are micro processors designed to perform DSP. Digital Signal Processing is one of core technology on the most rapidly growing areas such as wireless communication, industrial control, audio and video applications. With the rise in DSP processor applications there has been a huge increase in the DSP Processors in the industry since the 1980's.

If an application to be controlled or analyzed requires that time-critical functions be completed within a given time interval, then a system can be called a real-time system [16] if it will completely execute the necessary functions within the given time interval for all cases. More specifically, real time implies real time within the constraints of the system of interest . A real time DSP is a system which has to process and output data samples at the same frequency the input samples arrive.

Most DSP Processors are meant to handle high performance, repetitive, numerical task like the famous multiply and accumulate. Second important feature in DSP Processors is the ability to handle several memory accesses in the same instruction cycle like fetching instructions while fetching operands or storing the result of previous computations to memory. Third feature is the ability to generate address using the address generation circuitry in parallel with the other instructions though initially the address generation has to be configured and set. The DSP Processor has a loop counter implemented within so that there is no need for the programmer to expand the instruction cycle to implement a for loop. To facilitate low cost, high performance most DSP processors incorporate serial and parallel I/O interfaces, specialized I/O handling mechanisms like Direct Memory Access (DMA) or interrupt handler. This will allow data transfers to proceed with none or little intervention [16].

1.1.1 A Brief history of FPGAs

The FPGAs first arrived in the mid-1980s, they were largely used to implement glue logic medium complexity state machines, and relatively limited data processing tasks. During the early 1990s, as the size and sophistication of FPGAs started to increase, their big markets at that time were in the telecommunications and networking arenas, both of which involved processing large blocks of data and pushing that data around. Later, toward the end of 1990s, the use of FPGAs in consumer, automotive, and industrial applications underwent a humongous growth spurt [6].

FPGAs are often used to prototype ASIC designs or to provide a hardware platform on which the verification of the physical implementation of new algorithms was done. However, their low development cost and short time-to-market mean that they are increasingly finding their way into final products. By the early-2000s, high-performance FPGAs containing millions of gates has become available. Some of these devices feature embedded microprocessor cores, high speed-input/output (I/O) interfaces, and the like ones. The end result is that today FPGAs can be used to implement just about anything, including communications devices and software-defined radios, radar, image, and other DSP applications [5].

1.1.2 What can FPGAs be used for?

Mainly FPGAs are currently into four major market segments: Application Specific Integrated Circuit (ASIC) and custom silicon, DSP, embedded microcontroller applications, and physical layer communication chips. Furthermore, FPGAs have created a new market in their own right: Reconfigurable Computing (RC).

- **ASIC and custom silicon:** Today FPGAs are increasingly being used to implement a variety of designs that could previously have been realized using only ASICs and custom silicon.
- **Digital Signal Processing:** High speed DSP has traditionally been implemented using specially tailored microprocessors called Digital Signal Processors. However, the FPGAs available today can contain embedded multipliers, dedicated arithmetic routing, and large amounts of on-chip RAM, all of which facilitate DSP operations. When these features are coupled with the massive parallelism provided by FPGAs, the result is to outperform the fastest DSP chip by a factor of 500 or more.
- **Embedded microcontrollers:** Small control functions have traditionally been handled by special-purpose embedded processors called microcontrollers. These low-cost devices contain on-chip program and instruction memories, timers, and I/O peripherals wrapped around a processor core. FPGA prices are falling, however, and even the smallest devices now have more than enough capability to implement a soft processor core combined with a selection of custom I/O functions. The end result is that FPGAs are becoming increasingly attractive for embedded control applications.
- **Physical layer communications:** FPGAs are used to implement the glue logic that interfaces between physical layer communication chips and high-level networking protocol layers. As high-end FPGAs can contain multiple high-speed transceivers mean that communications and networking functions can be consolidated into a single device.
- **Reconfigurable Computing:** This refers to exploiting the inherent parallelism and reconfigurability provided by FPGAs to “hardware

accelerate” software algorithms. Various companies are currently building huge FPGA–based re- configurable computing engines for tasks ranging from hardware simulation to cryptography analysis.

1.2 Objectives of the Dissertation:

An attempt has been made to develop a simple 16-bit DSP Processor on FPGA. The instruction size, data path size as well as the operand size is of 16-bits.

The objectives of the dissertation are mentioned briefly:

- The individual blocks of the DSP Processor are implemented using VHDL (VHSIC Hardware Descriptive Language). These can be implemented using any HDL, there are two main languages namely Verilog and VHDL, In this dissertation, VHDL is used for the implementation. The algorithms implemented are then simulated and tested for suitability for synthesis. If these are not suitable for synthesis, the coding has to be done again. Care should be taken so that the HDL coding is done using the structural design method.
- The VHDL programs simulated are then synthesized on an suitable FPGA kit. In this work the Spartan 2, XC2S200 FPGA development kit is used. After the synthesis, the bit–streams are loaded onto the FPGA. This completes the implementation of DSP Processor on FPGA.

1.3 Organization of Thesis:

- Chapter 1 includes a brief introduction about Digital Signal Processing and Field Programming Gate Arrays. The various applications of DSP and FPGA is discussed along with the evolution and the applications in various fields.
- Chapter 2 explains about general DSP Processors and its environment, their architecture, applications.
- Chapter 3 explains about Instruction set architecture and addressing modes of the proposed processor.

- Chapter 4 gives an insight to the Processor design flow that has been followed in this thesis work. It is the main chapter that gives an idea about how exactly the Processor design was carried out.
- Chapter 5 summarizes the various results and discussions.
- Chapter 6 gives the Conclusions and Scope for Future work. This includes the concluding remarks and the other developments expected in the same field.
- The Appendix A and B explain about the various softwares and hardware kit details utilized for this dissertation. The softwares and the FPGA development kit are listed. Detailed explanation about each of them is included for a better understanding. As the market is on the rise, there are more and more new FPGA development kits coming into the market. It is going on in a such a pace that every month, a new FPGA development kit is being released by some or the other VLSI company.
- Appendix C gives the RTL Diagrams of the processor.

Chapter 2

DSP Processors

2.1 INTRODUCTION TO DSP PROCESSORS

2.1.1 DSP Environment

The Digital Signal Processing (DSP) Processor is designed specially for processing digital signals. DSP has a wide range of applications such as RADAR, Audio, SONAR, Telephony, Process Control, Digital Television, and Facsimile. There are a wide range of applications they have some common features like they have a lot of math involved, deal with signals from real world and require response in certain time [17,18].

The main advantages of using digital systems are repeatability, versatility and simplicity.

- Repeatability is when they can be easily duplicated; responses do not drift away with temperature and do not depend on strict component tolerances.
- Versatility is when systems can be reprogrammed for other applications or can be ported to other hardware like board level product. And finally
- Simplicity means some things are easier in digital systems than in analogue systems.

Most DSP applications deal with analogue signals which have to be converted to digital signals for processing as in Figure 2-1. During this conversion some information is lost due to uncertainty in timing, limits on duration of measurement or inaccuracies in measurement. The effects are called Quantization Errors. The continuous analogue signal has to be held before it can be sampled and measured so that value is not changing during the measurement. Then once it has been measured the signal will be converted to digital value for processing. The sampling results in a discrete set of digital numbers that were measured in equal intervals of time. The sampling takes places after the hold. So the Analog to Digital Converter (ADC) has to be fast

enough to sample the signal before the next signal value is taken in by the hold circuit. In short the ADC will have all the time to perform the conversion as long as the signal is held. A fast sampling needs to be done so that the rapid changes in the incoming signals are not lost. If not some higher frequencies can be interpreted as low frequencies [5,22].

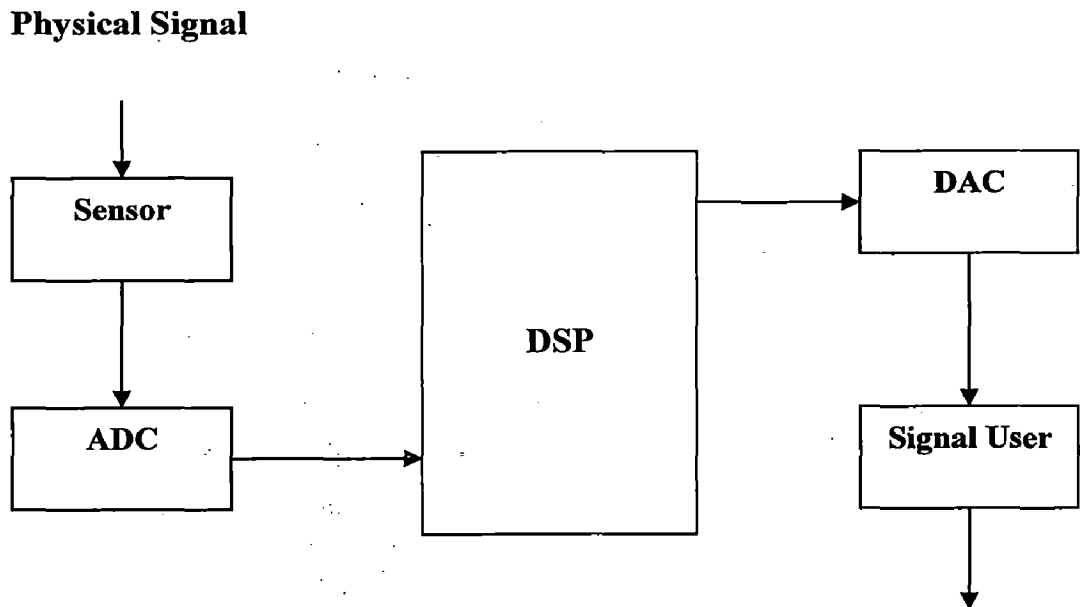


Figure 2.1. DSP System Overview

Physical Signal

2.1.2 Applications :

Applications of DSP are RADAR, Audio, SONAR, Telephony, Process Control, Digital Television, and Facsimile. DSP Processors have a wide range of applications from consumer electronics to Radar. There is no specific DSP Processor that will match all the applications. It is the designer choice to first evaluate the constraints in hand like power consumption, area, speed, performance, integration and other metrics in hand. In terms of money volume the biggest applications of DSP are in portable digital audio players commonly know as mp3 now days, mobile phones and disk drives where DSP is used for servo control. In these applications the integration and cost are paramount. For portable productions the battery consumption is very important and hence the focus is on power consumption.

2.1.3 DSP Architecture

All DSPs consist of several fundamental modules: a digital signal processing core to perform mathematical operations, memory to store data and program instructions. As a stored-program machine, the processor must be told what to do every clock cycle. Typically, a DSP fetches an instruction and some data from memory, operates on these, and then returns the manipulated data to storage. The way this is conducted is not the same for all processors. Two different architectures can be identified: Von Neumann and Harvard (see Figure 2.2 and 2.3).

The DSP application, in addition to the memory and peripheral configuration, usually governs the type of architecture employed [5,16].

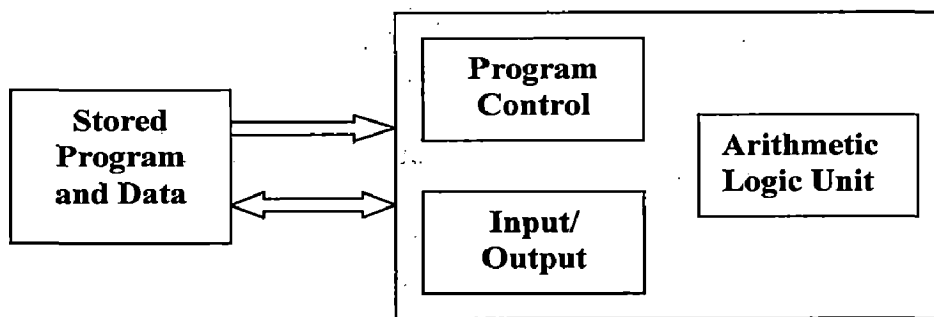


Figure 2.2. Von Neumann Architecture

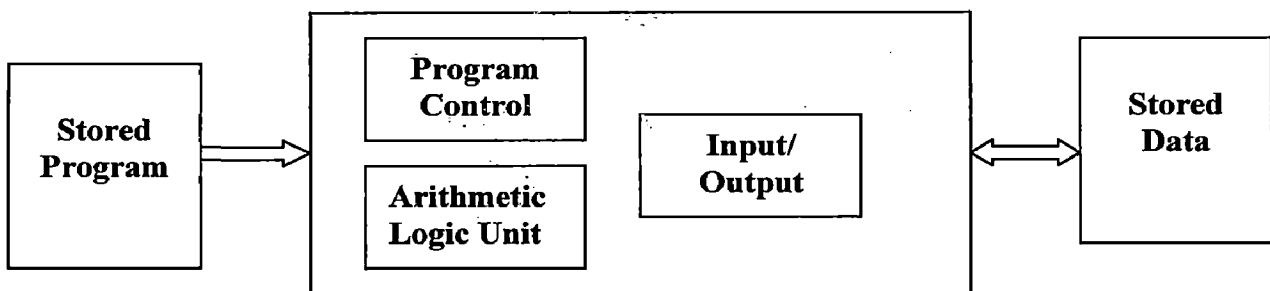


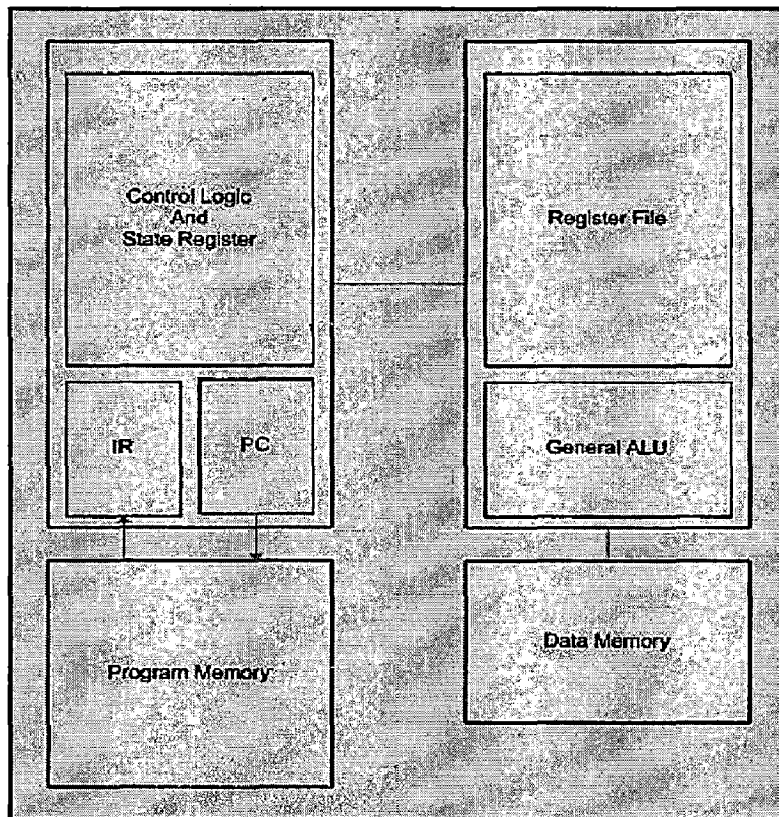
Figure 2.3. Harvard Architecture

Von Neumann architecture has set the standard for computer development over the past 40 years. Essentially, the architecture is very simple. Both program and data can reside in the same memory-mapped space. This architecture forms a basis for more general-purpose processing needs.

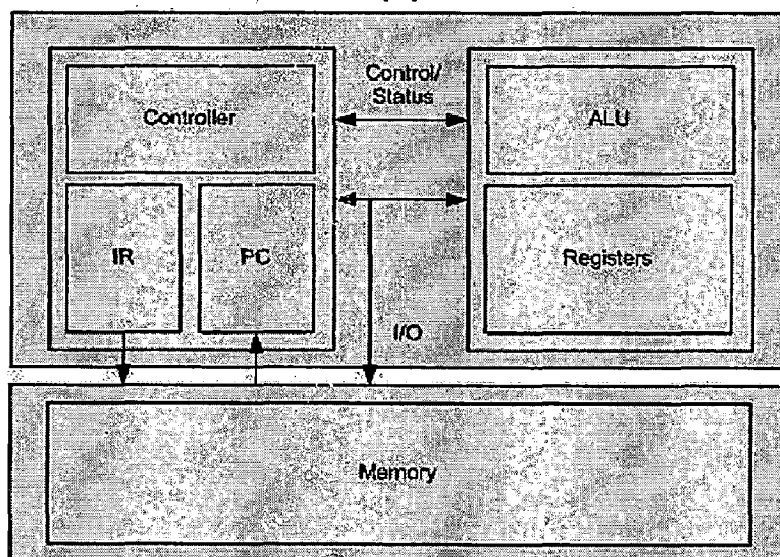
Harvard architecture separates the program and data memory spaces. Having two buses to serve each address space ensures that data and program access occurs in parallel, increasing processing speed.

2.2. General Architecture of DSP Processors

A DSP Processor is a programmable digital system intended to solve computational problems in a large variety of applications. A general purpose processor consists of a data path and a control unit tightly linked with a memory.



(a)



(b)

Figure 2.4 General Architecture of DSP Processor

All general DSP Processors Core is composed of the Data Path, Control Path and Program Path. The Memory Subsystem is located out of the processor core. Figure 2.3 (a) and (b) shows the general architecture of DSP Processor in both Harvard and Von Neumann fashion.

A basic DSP processor supports RISC and CISC instructions. The RISC uses the general registers for operands and writes them back to the Register File (RF). The CISC used the memory subsystem to compute vector elements like in the case of convolution. The CISC reads from the memory and write them to the Accumulator and to the registers.

The memory bus is distributed to the memory and DSP core DP components MAC and RF. However, there could be more components that can be connected to the memory bus. This depends on the choice of the instruction set which specifies all the operands required to perform a certain instruction. If there are instructions that ALU performs by fetching operands from the memory subsystem then the memory bus would also be connected to the ALU and so on. The Control Path generates the control signals for all components in the core, keeps track of the Program Counter and has a stack to service subroutines.

2.2.1 Data Path :

The datapath is the second main part of a processor. The datapath is responsible for the manipulation of data. The datapath consists of the circuitry for transforming data from and for storing temporary data. It includes (1) functional units such as adders, shifters, multipliers, ALUs, and comparators, (2) registers and other memory elements for the temporary storage of data, and (3) buses and multiplexers for the transfer of data between the different components in the datapath. External data can be entered into the datapath through the data input lines. Results from the computation are provided through the data output lines.

Processors are typically distinguished by their size, i.e. the bit width of the datapaths components. Common processor sizes include, 4-bit, 8-bit, 16-bit, 32-bit and 64-bit.

2.2.2 Control Unit :

Even though the datapath is capable of performing all the operations of the processor, it cannot, however, do it on its own. In order for the datapath to execute the operations automatically and correctly, the appropriate control signals must be asserted at the right time. In a processor, these control signals are generated by the control unit.

The control unit, also known as the controller, controls the operations of the datapath, and therefore, the operations of the entire processor. The controller is a finite state machine (FSM) because it is a machine that executes by going from one state to another, and the fact that there are only a finite number of states for the machine to go to.

The control unit consists of circuitry for retrieving program instructions and for moving data to, from and through the datapath according to those instructions. For each instruction the controller typically sequences through several stages, such as fetching the instruction from memory, decoding it, fetching operands, executing the instruction in the datapath, and storing the results. Each stage may consist of one or more clock cycles. A clock cycle is usually the longest time required for data to travel from one register to another.

2.2.3 Memory:

The registers serve a processor's short-term storage requirements, memory serves the processor's medium and long-term information-storage requirements, i.e. the storage information as either program or data. Program information consists of the sequence of instructions that cause the processor to carry out the desired system functionality. Data information requests the values being input, output and transformed by the program.

Chapter 3

Developed Processor: Instruction Set Architecture and Addressing Modes

3.1. Introduction :

The operation of the processor is determined by the instructions it executes, referred to as machine instructions. The collection of the different instructions that the processor executes is referred to as the instruction set. Each instruction is represented by sequence of bits. The instruction is divided into fields, corresponding to the constituent elements of the instruction [13,14,19].

The decision regarding the instruction format and addressing modes of a processor is the most important stage in the design. This decision will lead to the structure of the data path and the control unit. In the present work a 16 bit instruction format is adopted for the implementation of the proposed processor, where the operands and opcodes are of 16 bits. The instructions are of variable length. The instruction format is explained in detail in next few pages.

3.2 Instruction Format :

An instruction format must include an opcode and implicitly, zero or more operands. The format must, explicitly or implicitly, indicate addressing mode for each operand. The various fields of the instruction format are explained below.

Opcode	RD	RS	SHAMT	SE	I/Amode	I/Dcontrol
5 bits	3 bits	3 bits	2 bits	1 bit	1 bit	1 bit

Immediate Data

(2 Bytes/none)

Figure 3.1. Instruction format

The meaning of each of the fields is given as

- **Opcode:** Specifies the operation to be performed. It is the basic operation of the instruction.
- **RD:** The destination register operand, it gets the result of the operation.
- **RS:** Source register
- **SHAMT:** Shift amount
- **SE:** Shift enable bit
- **IAMode:** Indirect Addressing mode control bit
- **I/D control:** Increment/decrement after the operation when IAMode bit is 1.

3.3. Addressing modes:

The way of representing the operand in the instruction is known as the addressing mode. The proposed DSP Processor's instruction set provides the following addressing modes.

- **Register addressing modes:** The register will be identified in the instruction and the content of the register will be the operand. The address field that references registers is of 3 bits, thus 8 general purpose registers can be referenced. This addressing mode includes the data transfer instructions, i.e. the register to register data transfer instructions as well as the arithmetic instructions.
- **Immediate addressing:** It is the simplest form of the addressing, in which the operand is actually present in the instruction. This mode uses constants as operands to set the initial values of variables.
- **Direct addressing mode:** In this the operand is at the memory location whose address is specified in the instruction.
- **Indirect addressing mode:** In this the operand is at the memory location whose address is present in the register and the register is specified in the instruction.

3.4. Instruction types:

The instruction set is classified depending on the operands, broadly under the following categories.

- R-type instructions
- I-type instructions
- J-type instructions

3.4.1 R-type instructions:

The format for the R-type instruction is given in the figure.

Opcode	RD	RS	SHAMT	SE	Ignored	Ignored
5 bits	3 bits	3 bits	2 bits	1 bit	1 bit	1 bit

Figure 3.2 Fields in the R-type instruction

The operation which has to be performed on two registers is decided by the Opcode field. The resultant value of the operation will go to the destination register which is provided in the field named RD. The data path for the R-type instructions is shown below.

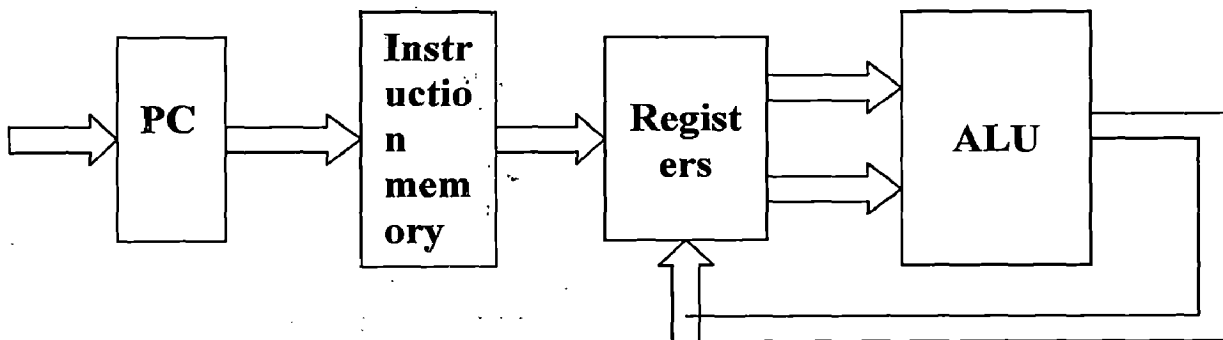


Figure 3.3 Data path for an R-type instruction

3.4.2 I-type instructions:

The instruction format of I-type instructions is shown in figure. Since the immediate data present in the instruction itself, this instruction needs another

word to contain the immediate data (either data or address). Some of the fields in the instruction are ignored.

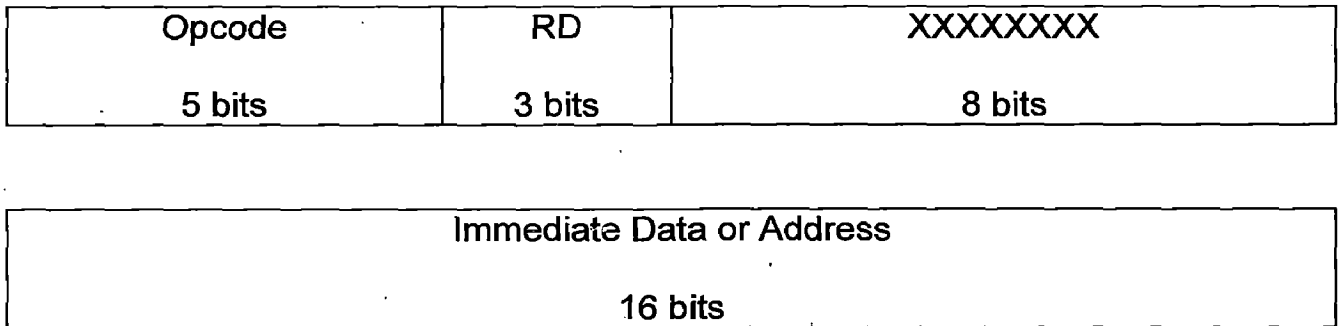


Figure 3.4 Fields in an I-type instructions

This group includes instructions with immediate operand, which is of 16 bits in length. The data path for immediate data instructions is shown in figure.

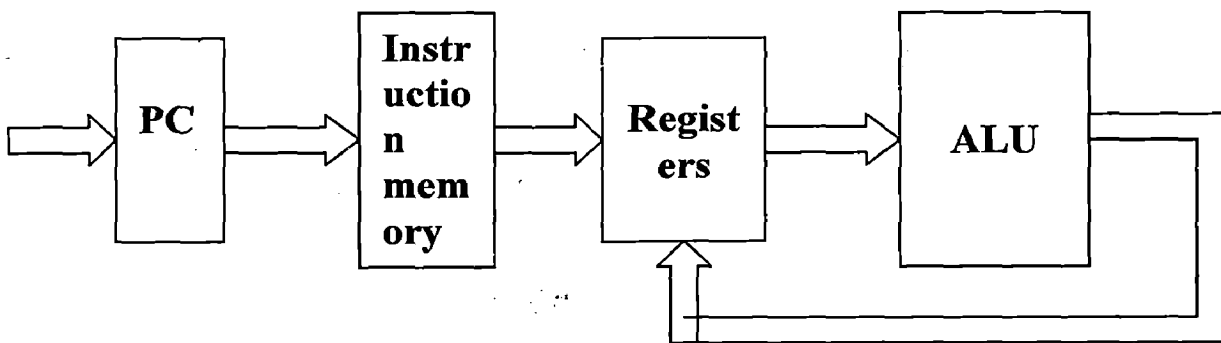


Figure 3.5. Data path for an I-type instruction

3.4.3 J-type instruction:

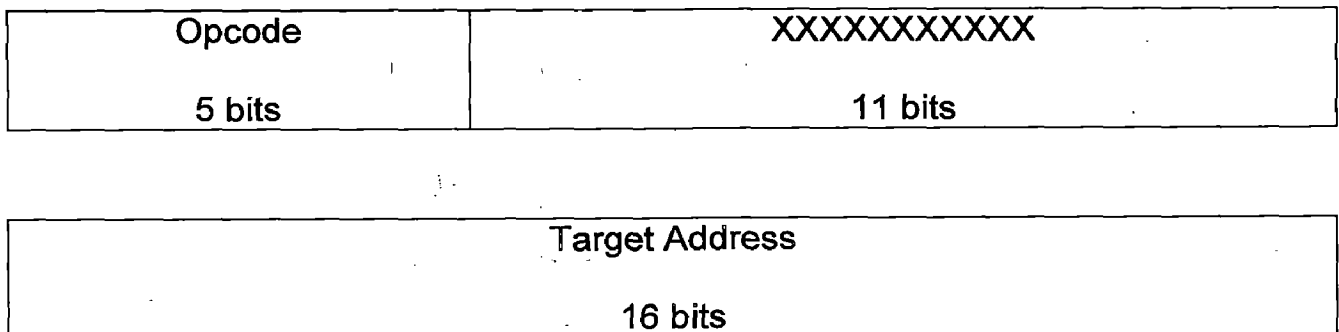


Figure 3.6 Instruction fields for Jump instruction

These instructions require 16-bit address field to specify the target of the jump. The instruction format is shown in figure. The target address is of 16-bits. So, the target can be any location within a range of 2^{15} locations.

The data path for the Jump instruction is shown in the figure 3.7. The necessary control signals are decoded in the decode stage and depending on these controls the next address for PC is decided.

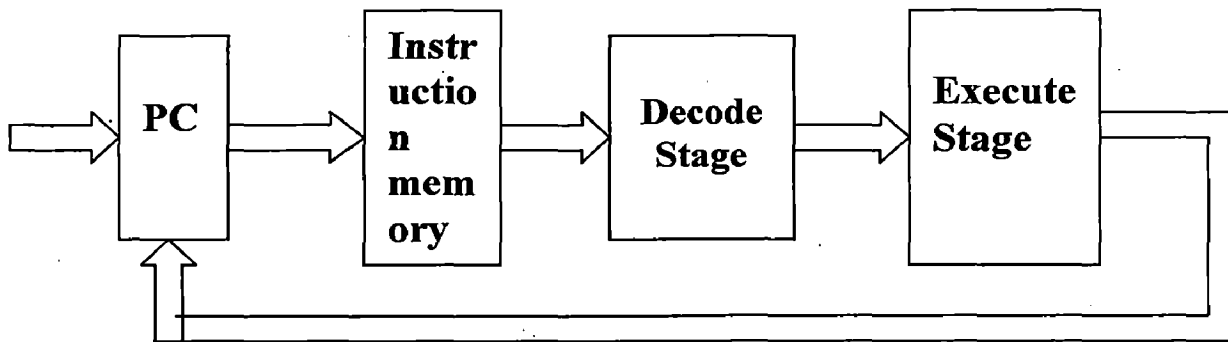


Figure 3.7 Data path for Jump instruction

3.5. Instruction set:

The total instruction set for the proposed DSP Processor consists of both single word and double word instructions. These instructions are classified into the following, as per the operation they perform as follows

- Data transfer instructions
- Arithmetic instructions
- Logical instructions
- T register, P register and multiply instructions
- Shift instructions
- Branching instructions

3.5.1 Data transfer instructions:

MVI R, IMMDATA

The immediate data of 16 bits is loaded into the register specified as R. The opcode for the instruction is of the format

OPCODE	R VALUE	XXXXXXXX	IMMDATA
5 Bits	3 Bits	8 Bits	16 Bits

MOV Rd, Rs

The data in Rs is copied to Rd. The opcode for the instruction is of the format

OPCODE	R VALUE	R VALUE	XXXXX
5 Bits	3 Bits	3 Bits	5 Bits

MOV Rd, (Rs)

The data in the address specified by the register Rs is copied in to Rd. The opcode for the instruction is of the format

OPCODE	R VALUE	R VALUE	XXX	1	I/D
5 Bits	3 Bits	3 Bits	3 Bits	1 Bit	1 Bit

From the above format IAmode field '1' indicates the indirect addressing mode and I/D field indicates either the auto decrement or increment in Rs value.

LDAI IMMDATA

The accumulator is loaded with the IMMDATA. Accumulator is the implicit operand. The opcode for the instruction is of the format

OPCODE	XXXXXXXXXXXX	IMMDATA
5 Bits	8 Bits	16 Bits

STORE (Rd), Rs

The data in register Rs is copied in to the address specified in the register Rd. The opcode for the instruction is of the format

OPCODE	R VALUE	R VALUE	XXX	1	I/D
5 Bits	3 Bits	3 Bits	3 Bits	1 Bit	1 Bit

3.5.2 Arithmetic instructions:

INC/DEC R

The data in register R is either incremented or decremented as per the operation and the result is stored in the same register.

OPCODE	R VALUE	XXXXXXXX
5 Bits	3 Bits	8 Bits

ADD/SUB Rs

The data in the register Rs is operated with the accumulator as per the instruction and the result is stored in the accumulator. The instruction format is shown below.

OPCODE	XXX	R VALUE	XXXXX
5 Bits	3 Bits	3 Bits	8 Bits

3.5.3 Logical instructions:

ORA/ANA/XRA Rs

The value of Rs is operated as per the logical operation with the contents of accumulator and the result is stored in the accumulator. The instruction format is shown below.

OPCODE	R VALUE	XXX	XXXXXXXX
5 Bits	3 Bits	3 Bits	8 Bits

ORA/ANA/XRA IMMDATA

The content of accumulator and immediate data are operated according to the logical operation specified and the result is stored in the accumulator. The instruction format is shown below

OPCODE	XXXXXXXXXXXX	IMMDATA
5 Bits	11 Bits	16 Bits

3.5.4 T register, P register and multiply instructions:

LT IMMDATA

The T register is loaded with the sign extended form of the IMMDATA. The IMMDATA is sign extended to 16 bits and is loaded into T register. The opcode for the instruction is of the format

OPCODE	IMMDATA
5 Bits	11 Bits

MPY R

The contents of the register R are multiplied with the contents of implicit operand T register and the result is placed in the P register. The instruction format is shown below

OPCODE	XXX	R VALUE	XXXXX
5 Bits	3 Bits	3 Bits	5 Bits

3.5.5 Shift instructions

SHL/R Rd, Rs, SHAMT

The shift instruction shifts the data present in the register Rs by number of bits specified in the field SHAMT and stores the result in register Rd. The instruction format is shown below

OPCODE	R VALUE	R VALUE	SHAMT	SE	XX
5 Bits	3 Bits	3 Bits	2 Bits	1 Bit	2 Bits

3.5.6 Branching instructions

JMP IMMDATA

The jump instruction always branches to the target address that is specified in the instruction itself. The instruction format is shown below

OPCODE	XXXXXXXXXXXX	IMMDATA
5 Bits	11 Bits	16 Bits

Chapter 4

Design Architecture of DSP Processor

4.1 Introduction

The circuit for the DSP Processor can be divided into two parts: the datapath and the control unit. The datapath is responsible for the actual execution of all operations performed by the DSP Processor. The control unit, also known as the controller, provides all control signals to the programPath and dataPath.

Every digital logic circuit, regardless of whether it is part of the control unit or the datapath, is categorized as either a combinational circuit or a sequential circuit. A combinational circuit is one where the output of the circuit is dependent only on the current inputs to the circuit. A sequential circuit, on the other hand, is dependent not only on the current inputs but also on all the previous inputs. Since sequential circuits are dependent on the history, they must therefore contain memory elements for remembering the history, whereas, combinational circuits do not have memory elements [10,14,19].

This chapter describes the various modules of the Processor Data Path, their design considerations and architecture. This chapter also describes about the ControlPath which is the most complicated and tricky in designing a Processor.

4.2 Architecture Overview

The architecture of the designed 16-bit DSP Processor is split into 3 main modules: controller, dataBus, and programBus. The controller consists of a finite state machine with states for instruction fetch and decode based on the 16-bit instruction set. The controller outputs control signals to every submodule in the dataBus and programBus. The program Bus module consists of a 16-bit bus called programBus (actually, Instruction Register), which contains each fetched

instructions from the Program ROM. The data Bus module consists of a 16-bit bus called dataBus, which has complete data information. The dataBus also contains an ALU, multiplier, accumulator, barrel shifter, and muxes [2,3,15]. The DSP Processor supports three addressing modes: direct, register-indirect, and immediate modes. The internal datapath architecture is shown in Figure 4.1.

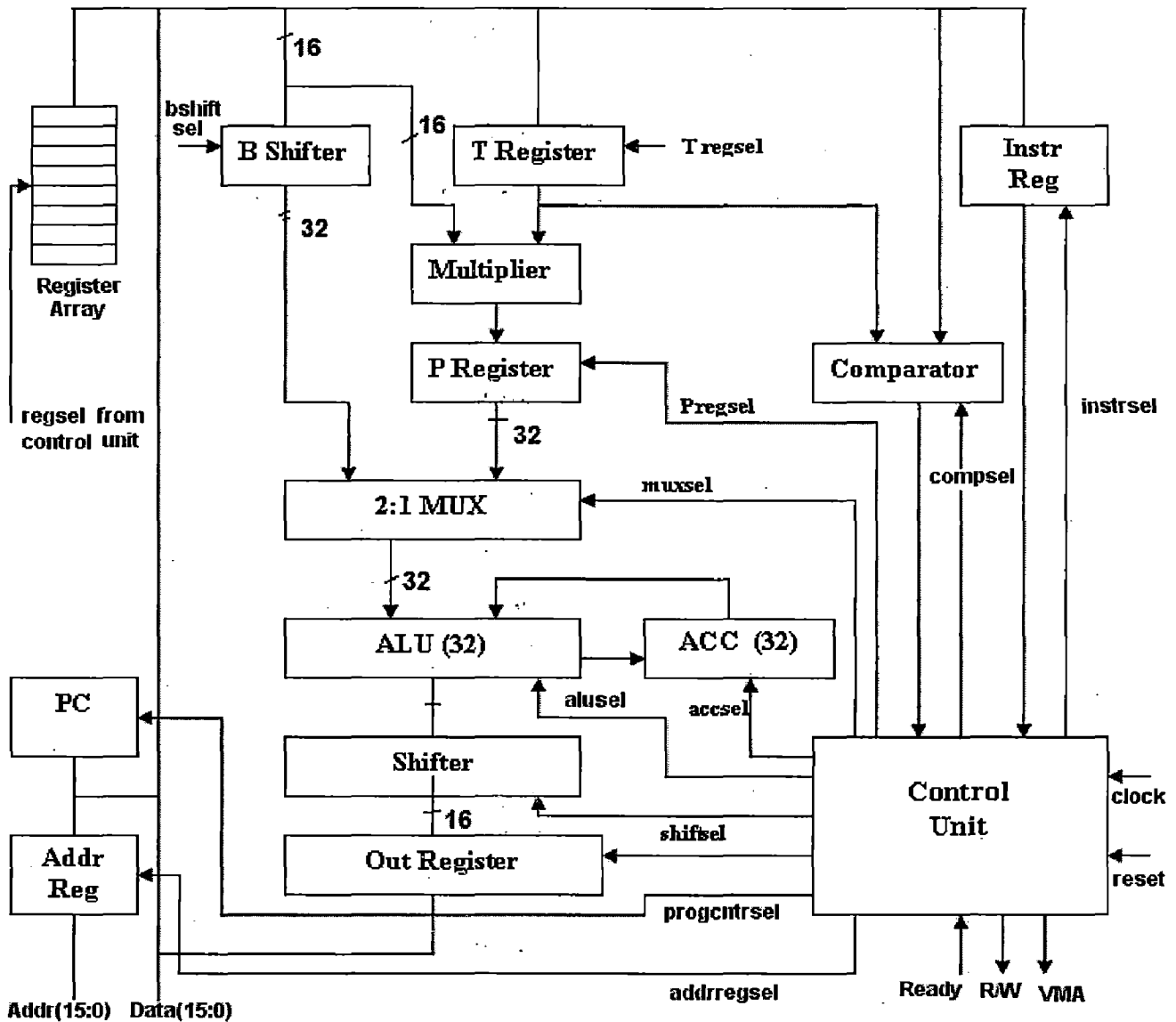


Figure 4.1. Internal Architecture of designed DSP Processor

The proposed processor's DataPath houses all the arithmetic hardware, a 32-bit ALU, 32-bit accumulator, and 16x16-bit multiplier with 32-bit product. It also contains a 144x16-bit data RAM. Two shifters are also used in the architecture. The 0 to 16 bit barrel shifter with 16-bit data bus input outputs a 32-bit value to the ALU. The second shifter takes the output of the accumulator and shifts 0, 1,

or 4-bits and outputs 16-bits to the dataBus. The register array (RF) used for indirect addressing and for storage of operands is also located within the dataPath. All these functional units inside the datapath (ALU, accumulator, shifter, multiplier, etc.) and the registers are connected together with multiplexers and buses to form one unit, the datapath. The 16-bit dataBus, which holds data to and from the data RAM and accumulator.

The controller provides all control signals to the programPath and dataPath. They include all the select lines for multiplexers, ALU and other functional units having multiple operations, all the read/write enable signals for registers and register files, address lines for register files, and enable signals for tri-state buffers. The operation of the datapath is determined by which control signals are asserted and at what time.

During startup and reset, an initialization state followed by two wait states set up the CPU. This allows initialization values to propagate through the logic. The decode state determines which instruction is on the programBus. Direct/indirect addressing functionality are also detected within this state. When an instruction execution is complete, the state machine will always return to the fetch/decode state.

The number of states per instruction varies from four to six states depending on the complexity of the instruction. The controller outputs mux controls, ALU functions, register reads/writes and other control signals to the DSP hardware depending on the current instruction. The program counter (PC) is enabled during the second to last instruction state. It takes two state cycles for the next instruction to stabilize on the programBus.

In return, the datapath needs to supply status signals back to the control unit in order for it to operate correctly. These status signals are usually from the output of comparators. The comparator tests for a given logical condition between two values. These values can be obtained either from memory elements, directly from the output of functional units, or hardwired as constants. These status signals provide input information for the control unit to determine what operation to perform next. For example, in a conditional loop situation, the status signal will tell the control unit whether to repeat or exit the loop.

4.3 Data Path (DP)

The datapath performs all the functional operations of a processor, and the processor is for solving problems, therefore the datapath must be able to perform all the operations required to solve the given problem.

Datapath design is also referred to as the register-transfer level (RTL) design. The register-transfer level design tells how data is transferred from one register to another or back to the same register. If the same data is written back to a register without any modifications, then nothing has been accomplished. So before writing the data to a register, the data passes through one or more functional units and gets modified.

4.3.1 Arithmetic and Logic Unit (ALU)

As the name describes, the ALU does all the logic and arithmetic computations. 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 4.2. All of the other elements of the processor bring data into the ALU [7,9,10,19] for it to process and then take the results back out.

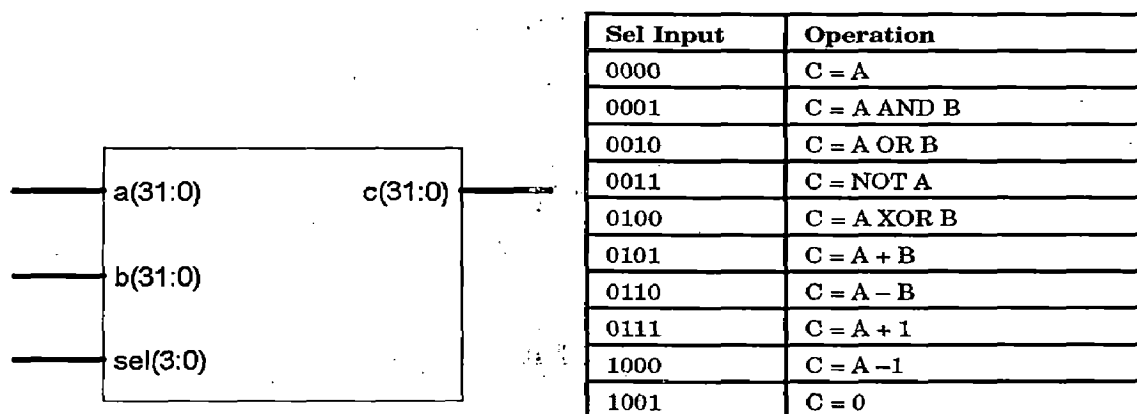


Figure 4.2 ALU

The proposed 16 bit DSP processor contains two 32-bit input ports and one 32-bit output port feeding the accumulator. ALU and accumulator for arithmetic operations. The ALU is a general purpose arithmetic unit; operations

performed with the 16-bit words taken from memory, the 16-bit words derived from immediate instructions, or the 32-bit result taken from the product register of the multiplier. In addition to the normal arithmetic instructions, the ALU perform Boolean operations, providing a bit manipulation ability required of a high speed controller.

Inputs **a** and **b** are the two input busses upon which the ALU operations are performed. Output bus **c** returns the result of the ALU operation. Both the inputs **a** and **b** and output **c** are of 32-bits. Input **sel** determines which operation is to be performed as specified by the table 4.1. The input **sel** is of 4-bits.

The control unit provides **sel(3:0)** signals that control the operation of the ALU and the movement of the data into and out of the ALU. The ALU can perform a number of arithmetic operations, such as add and subtract, and some logical operations, such as AND, OR, and XOR.

4.3.2 Comparator

This entity compares two values and returns either a '1' or '0' to the control unit as the status signals, depending on the type of comparison requested and the values being compared. Based on this status signal the controller determine what operation to perform next. A symbol showing the ports of the comparator is shown in Figure 4.3. The comparison type is determined by the value on input port **sel(2:0)**.

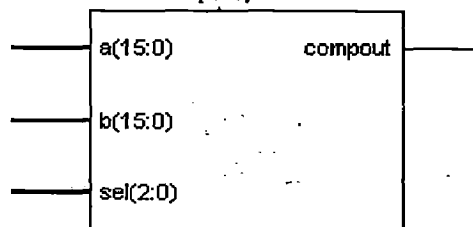


Figure 4.3. Symbol of Comparator

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. 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. If the condition tested is true, a '1' value is assigned; otherwise, a '0' is assigned. Figure 4.4 shows the internal architecture of comparator.

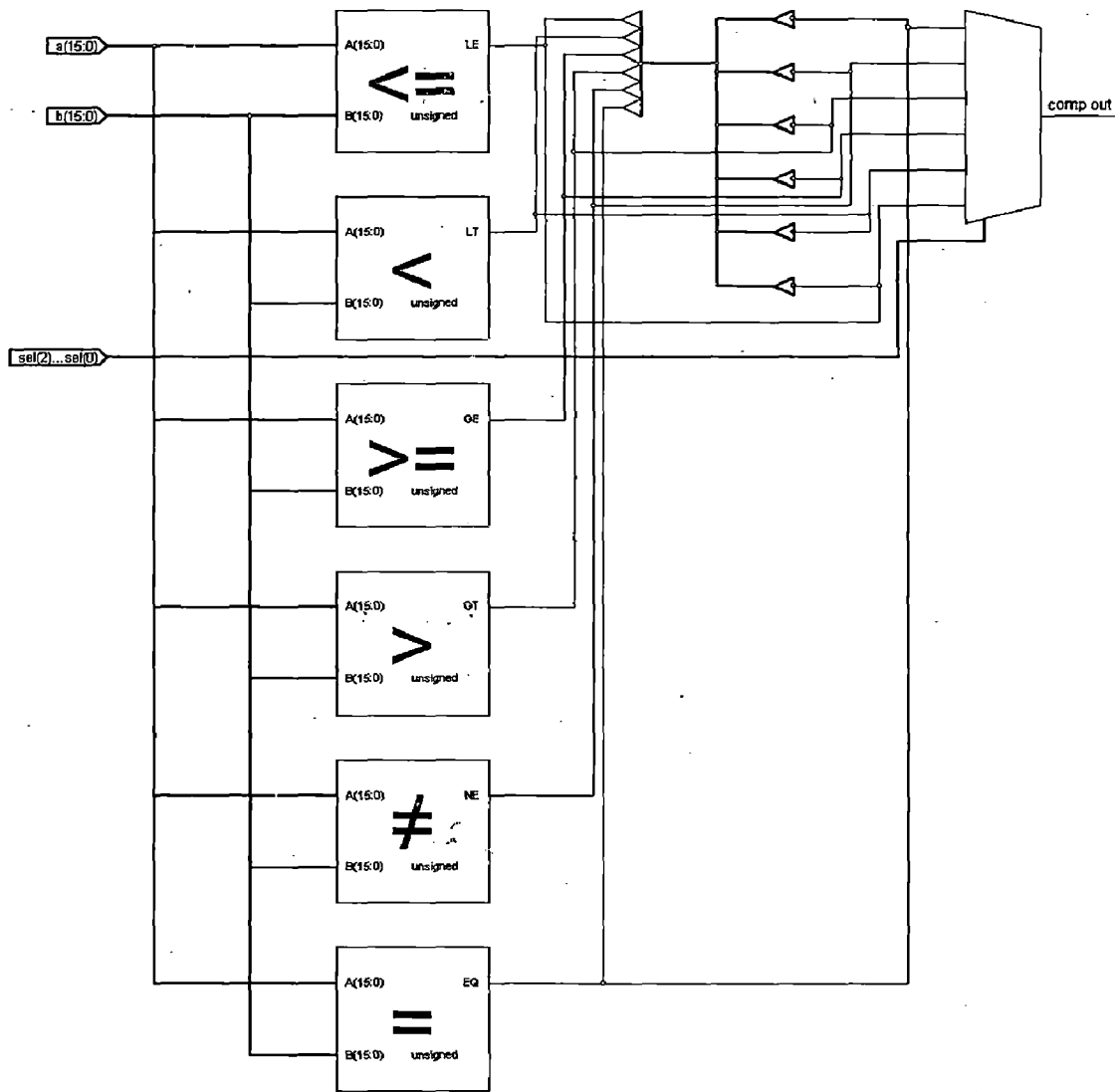


Figure 4.4. Architecture of comparator

4.3.3 Multiplier: Multiplication is one of the basic and critical operations in the computations. Efficient implementations of multipliers are required in many applications. In this present work a 16-bit array multiplier [4,5] is designed, whose block diagram is shown in figure 4.5. The multiplier performs a 16 x 16 bit multiplication with a 32-bit result in a single machine cycle. The multiplier consists of three elements:

- The T Register,
- The P Register,
- The multiplier array

The 16-bit T register temporarily stores the multiplicand; the P register stores the 32-bit product. The multiplier values either come from the data memory or are derived from instruction word. The on-chip multiplier allows the device to perform fundamental DSP operations such as convolution, correlation, and filtering.

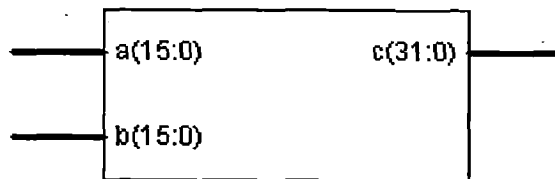


Figure 4.5. Multiplier Block diagram

There are 3 methods for performing the multiplication.

1. Serial/Parallel multiplication: In this one operand is used in parallel and the second operand is used bitwise, i.e. serial.

2. Serial/Serial multiplication: If both operands are used serial, the scheme is called a serial/serial multiplier. Such a multiplier only needs one full adder, but the latency of serial/serial multipliers is high, because the state machine needs about $N \times N$ cycles.

3.Parallel/Parallel or Array multiplication: In this approach both operands are used in parallel. It improves the speed of the operation with increased complexity. This arrangement is viable if the time required to complete the carry and sum calculations are same.For modern FPGA the carry computation is performed faster than the sum calculation and a different architecture is more efficient for FPGAs. The approach for this array multiplier is shown in figure 4.6.

This is a direct array form of “pencil and paper” method and must therefore produce a valid product. In this work, the array multiplier for unsigned numbers is

proposed. The proposed scheme is applicable for VLSI and FPGA application. For VLSI implementations of multipliers, array-based multipliers are well known and often used.

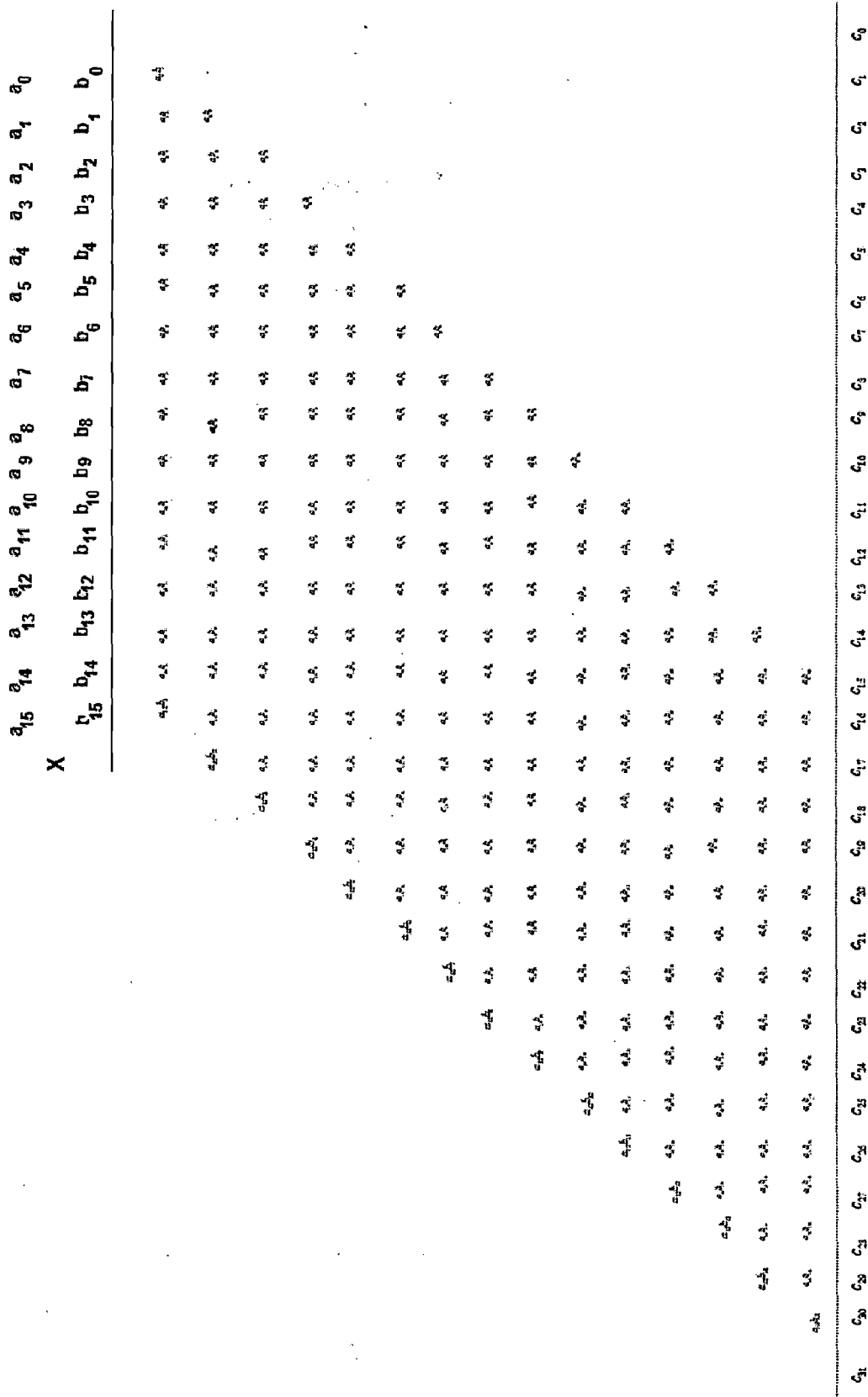


Figure 4.6 A 16-bit Array Multiplication

4.3.3.1 Array-Based Multiplication:

An array multiplier was first proposed in [1,4,5] which has good repeatability of unit cells and is very regular in its structure. It uses only short wires that connect one full adder horizontally, vertically, or diagonally adjacent full adders. Thus, it results in a very simple and efficient layout in VLSI implementation.

The operation of an 16-bit multiplication is shown in figure 4.9, where the two operands are $a(15:0)$ and $b(15:0)$ and the out put is $c(31:0)$. The operation contains all 256 partial product bits of the form $(a_i \times b_j)$ properly aligned. The partial product in the first row is added to that in the second row, and so on. The final product is thus obtained.

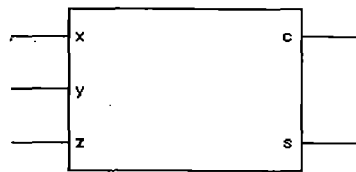


Figure 4.7. Full Adder symbol

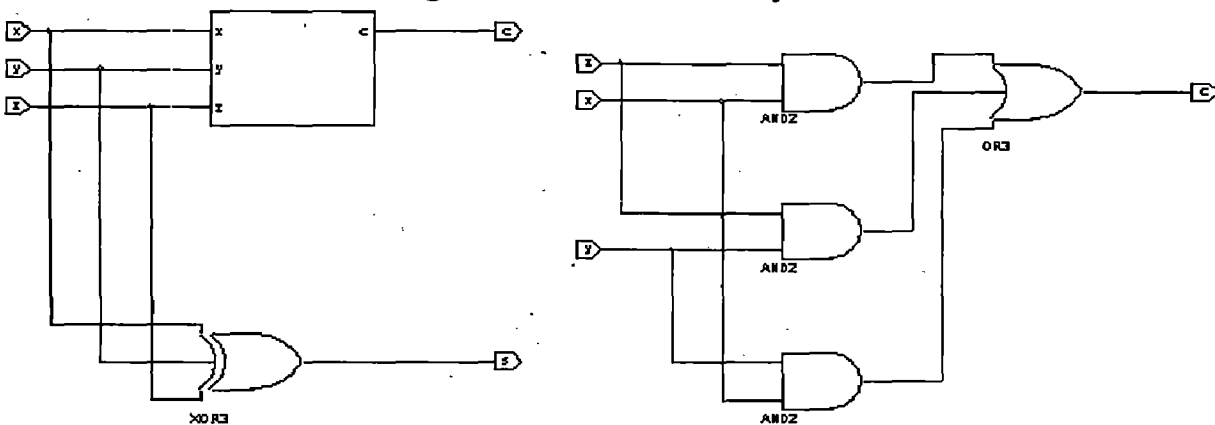


Figure 4.8. Full adder RTL diagram

The N-bit Array multiplier, requires $N \times N$ two input AND gates and $N \times (N-1)$ Full-Adders. The 16 bit array multiplier requires 256 two input AND gates and 240 Full-Adders. The same principle is applied to 32 bit, 64 bit array multipliers.

To use the multiplier, first one of the operand must be loaded into the T register by load instruction. Then by multiply immediate instruction, the second operand will be loaded, thus the multiplier computes the multiplication of those two operands. The result of the multiplier is stored in the P register.

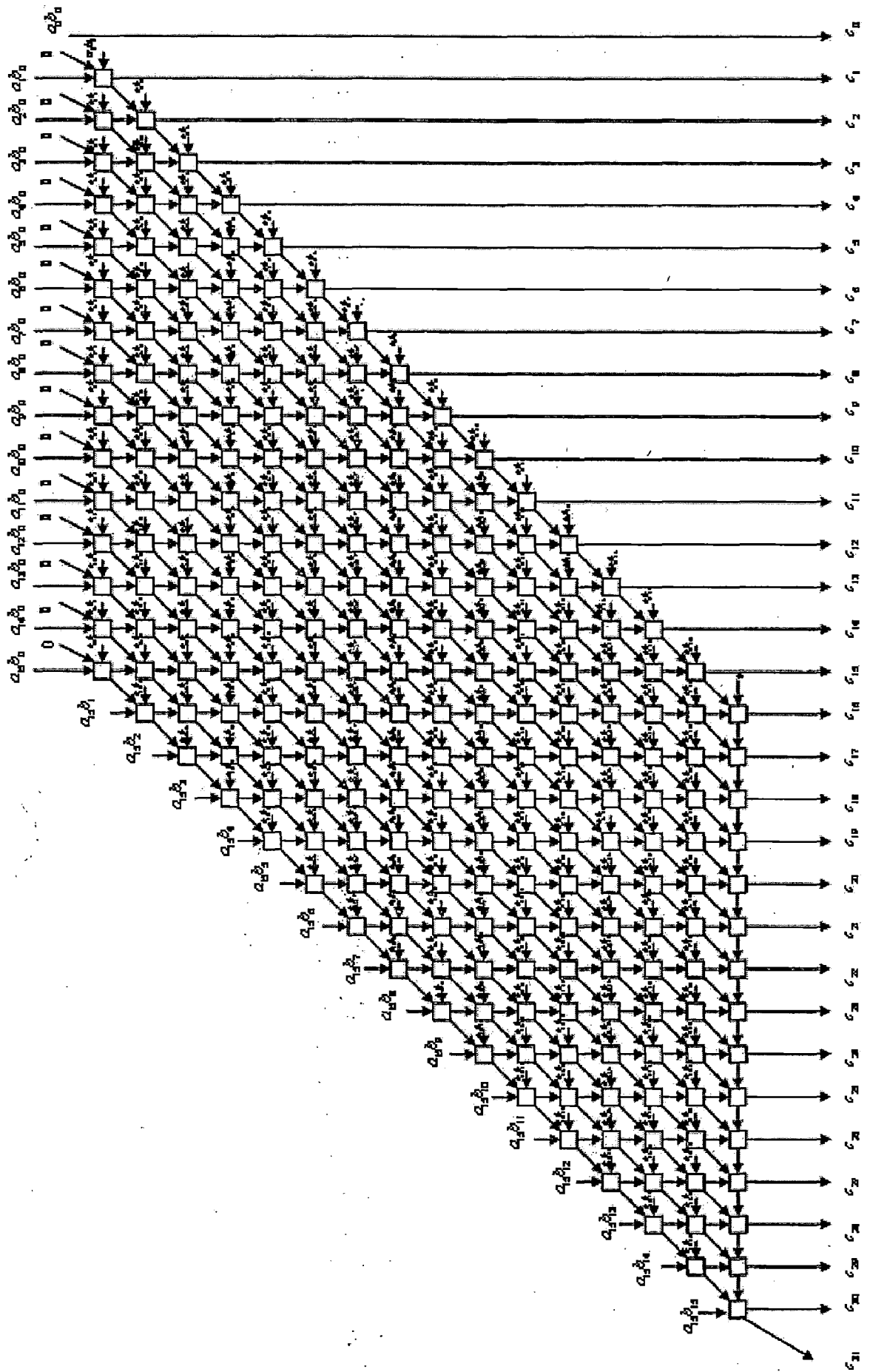


Figure 4.9. A16 Bit-Array Multiplier

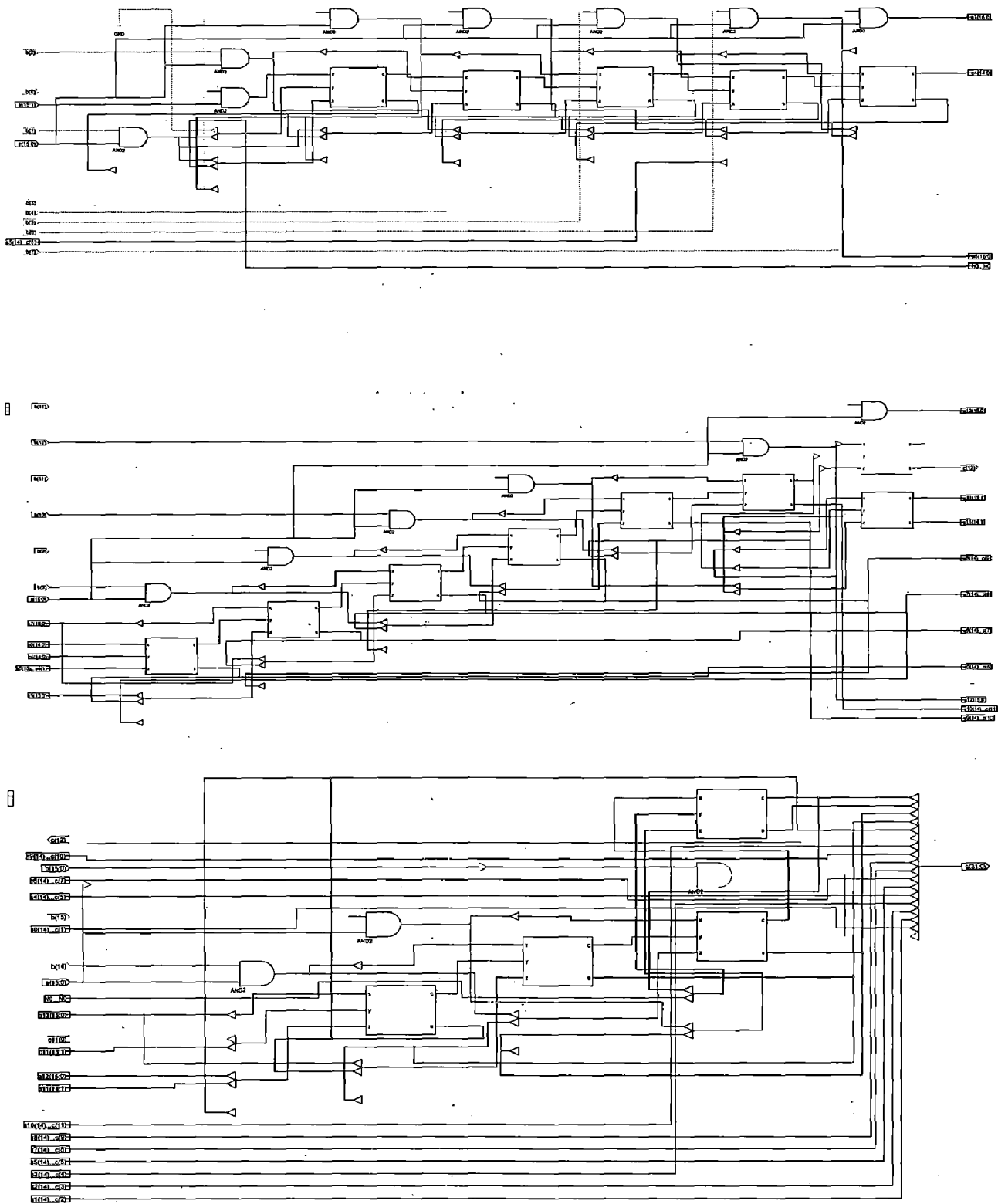


Figure 4.10. Multiplier Internal Architecture

The multiplier architecture that is generated in Xilinx ISE 7.1 tool is shown in figure 4.10. This consists of full adders and the AND gates that are required in the construction of the multiplier.

4.3.4 Register File:

The Register File (RF) is a group of general purpose registers used as a storage buffer. This register array or Register File [9,10,19] 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 4.11.

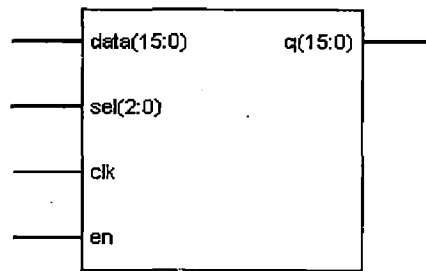


Figure 4.11. Register File Symbol

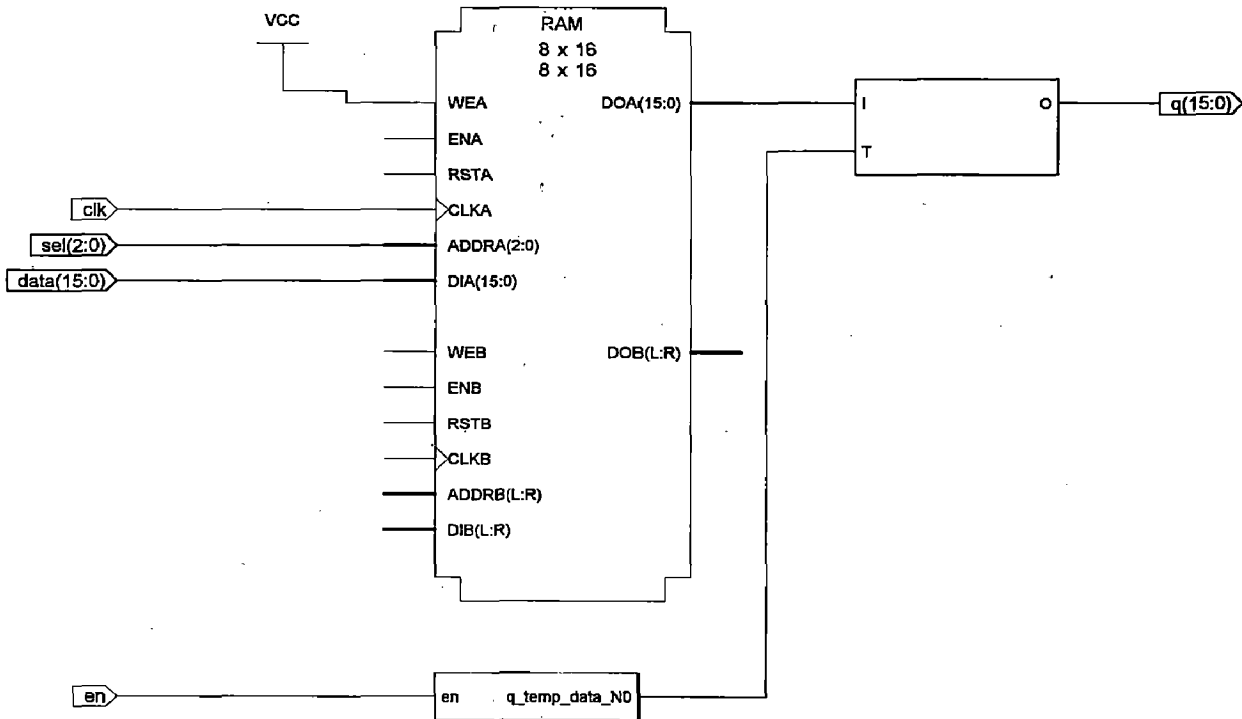


Figure 4.12. Register File internal architecture.

From the fig 4.12. it shows two register arrays A and B. Each array consists of 8 registers which are of 16-bits each. The individual register array is used by the application of corresponding signals. In this present work register array A is used. To write a location in the regarray, set input **sel** to the location to be written, input data with the data to be written, and put a rising edge on input

clk. To read a location from regarray, set input sel to the location to be read and set input **en** to a '1'; the data is output on port q. When the **clk** signal has a rising edge, the location selected by input sel is updated with the new value.

4.3.5 Multiplexer:

The multiplexer connects multiple inputs to a single output. At any time, one of the inputs is selected to be passed to the output. In the present work a 2-to-1 multiplexer(MUX) is developed. The block diagram representation of 2-to-1 MUX is shown in figure 4.13.

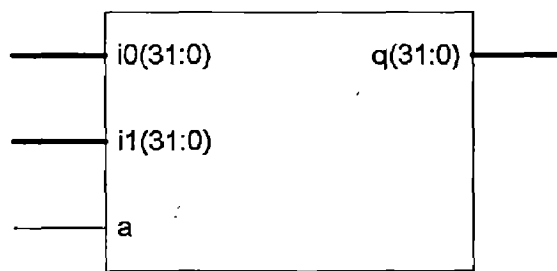


Figure 4.13. Block diagram representation of 2-to-1 MUX

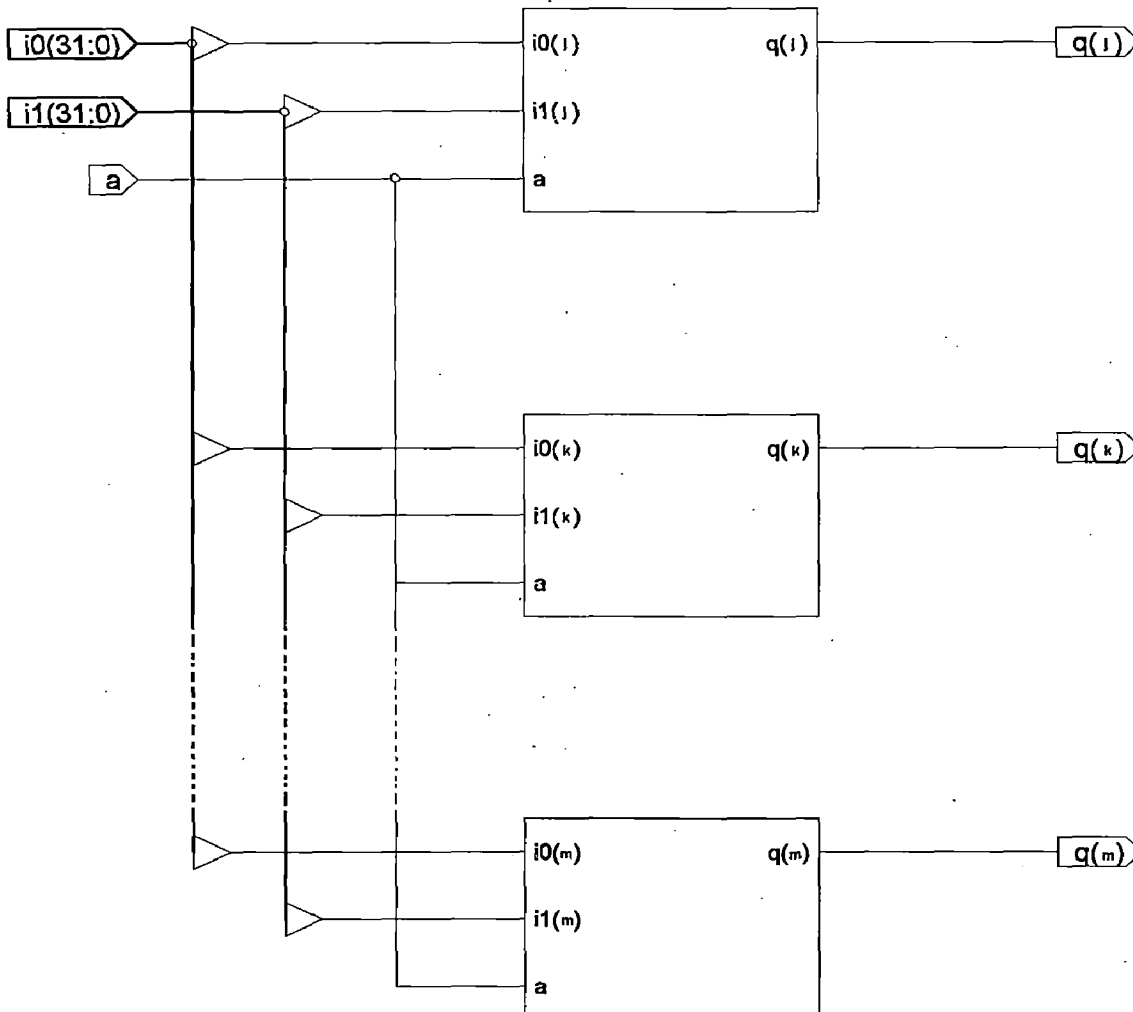


Figure 4.14. Multiplexer

There are two input lines ,labeled i_0 and i_1 , both are 32 bit length. One of these lines is selected to provide the output q . To select one of the two possible inputs, a selection line needed, and this is implemented as a . At any time the output is either i_0 or i_1 for MUX selection line a is at '0' or '1', respectively.

Even though the total input line gets selected to the output line, the actual selection is based on the bit by bit basis as shown in the figure 4.14.

4.3.6 Shifters:

Two shifters are available for manipulating data: a Barrel shifter for shifting data from data RAM into the ALU and a parallel shifter for shifting the accumulator into data RAM.

Barrel shifter: A barrel shifter [7,9] is a digital circuit that can shift a data word by a specified number of bits. The barrel shifter has a 16-bit input connected to the data bus and a 32-bit output connected to the ALU.

This shifter extends the high-order bit of the data word and zero-fills the low-order bits. The barrel shifter produces a left shift of 0 to 16 bits on all data memory words that are loaded into the accumulator. The shifter sign extends the 16-bit data memory word to 32 bit by an arithmetic left shift operation. Simultaneously logical shift operations can also be performed.

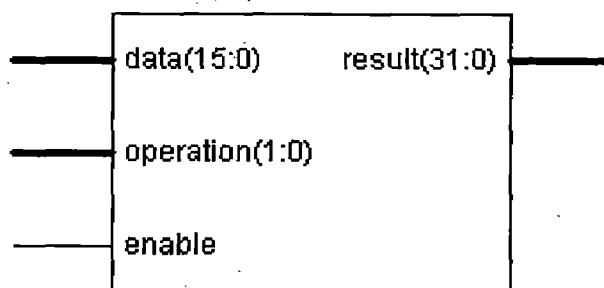


Figure 4.15. Block diagram of Barrel Shifter

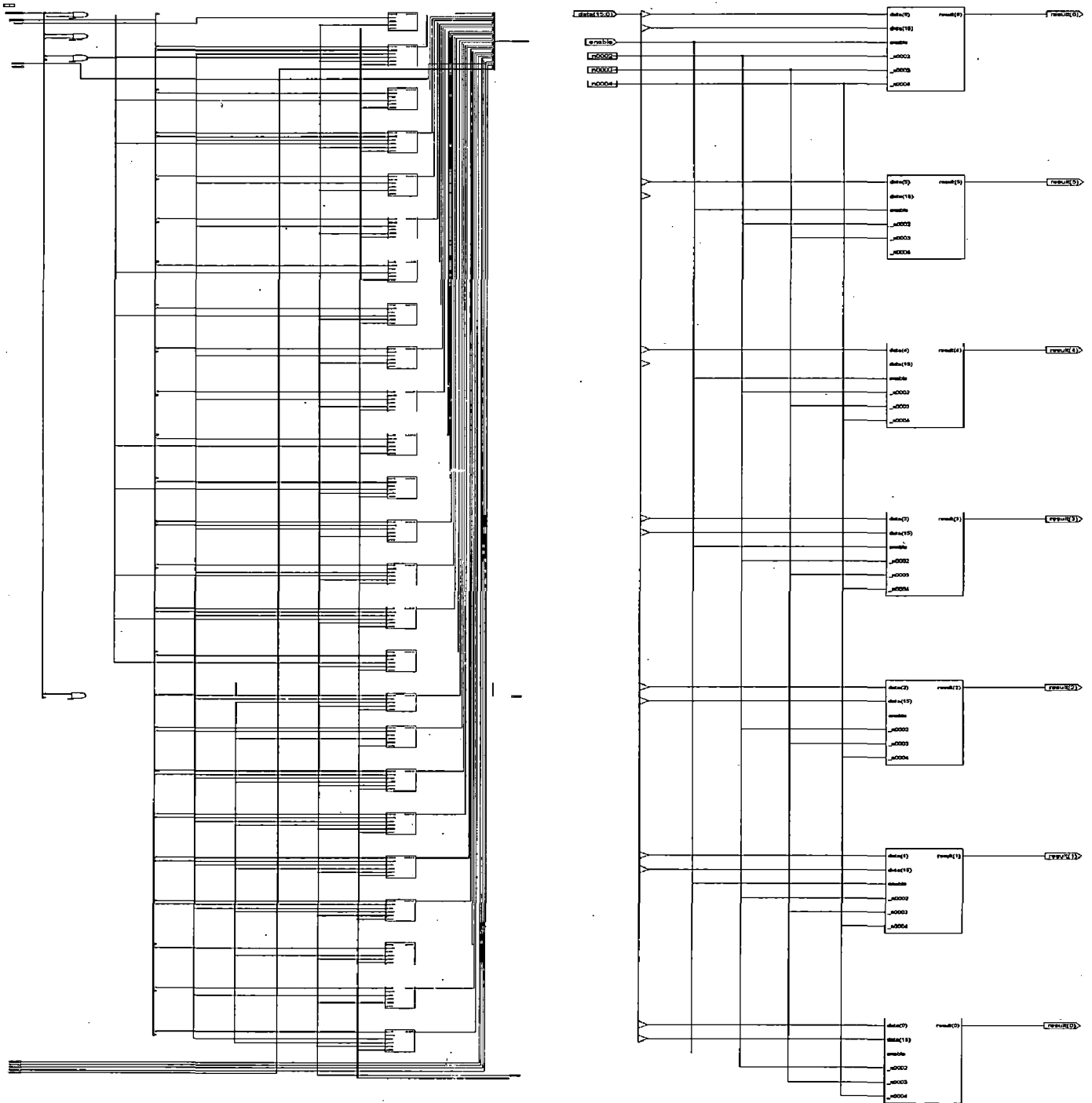


Figure 4.16 RTL Diagram of Barrel Shifter

Parallel shifter:

The accumulator parallel shifter performs a left-shift of 0,1 or 4 places on the entire accumulator and places the resulting high-order bits into memory. The Parallel shifter is loaded with the 32-bit content of the accumulator. The data is then left-shifted. The most significant 16 bits from the shifter are stored in RAM, resulting in a loss of remaining 16-bits. The contents of the accumulator remain unchanged.

4.3.7 Reg

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 4.17.

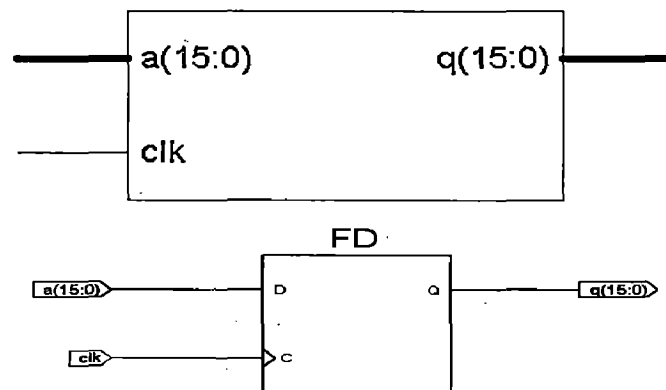


Figure 4.17. Register Symbol and RTL diagram

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. The same architecture is for the 32 bit registers, whereas the inputs and output are of 32 bits.

4.3.8 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 **triereg** entity has four ports as shown in Figure 4.18.

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. When a rising edge is applied to input **clk**, the data on input **a** is stored in the register. Input **en** is used to control output **q**. When **en** is a '1' value, the register state is driven to output **q**. When **en** is a '0', output **q** is a high impedance value and not driving.

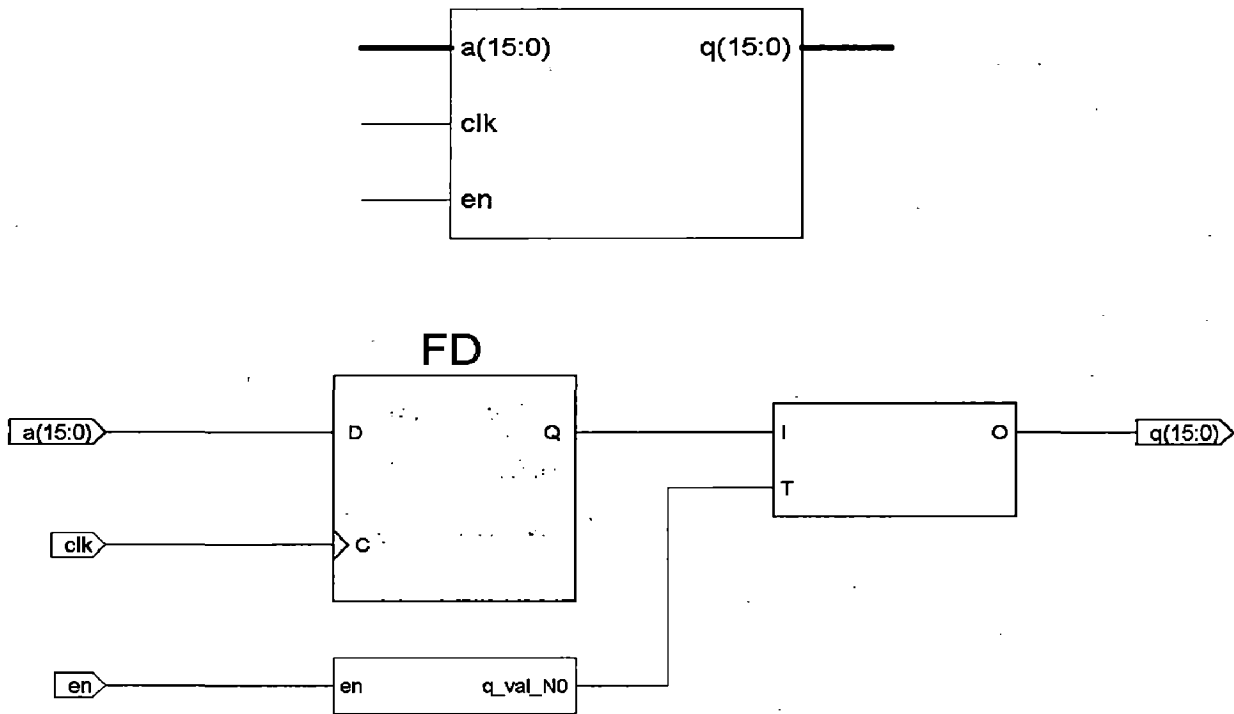


Figure 4.18 Tri Register Symbol and RTL diagram

4.4 Controller:

The control unit [7,9,19] is a sequential circuit in which its outputs are dependent on both its current and past inputs. This history of past inputs is stored in the state memory and is said to represent the state of the circuit. Thus, the circuit changes from one state to the next when the content of the memory changes. Depending on the current state of the circuit and the input signals, the next-state logic will determine what the next state ought to be by changing the content of the state memory. Hence, a sequential circuit executes by going through a sequence of states. Since the state memory is finite, therefore the total number of different states that the circuit can go to is also finite.

Finite-state machines are classified into two main types: Moore and Mealy. A Moore type FSM is one where the output of the machine is dependent only on the current state, whereas a Mealy type FSM is one where the output is dependent on both the current state and the input signals.

The control unit (or controller) is responsible for controlling all the operations of the datapath by providing appropriate control signals to the datapath at the appropriate time. At any one time, the control unit is said to be in a certain state as determined by the content of the state memory as shown in

figure 4.19. The state memory is simply a register with one or more (D) flip-flops. The control unit operates by transitioning from one state to another – one state per clock cycle.

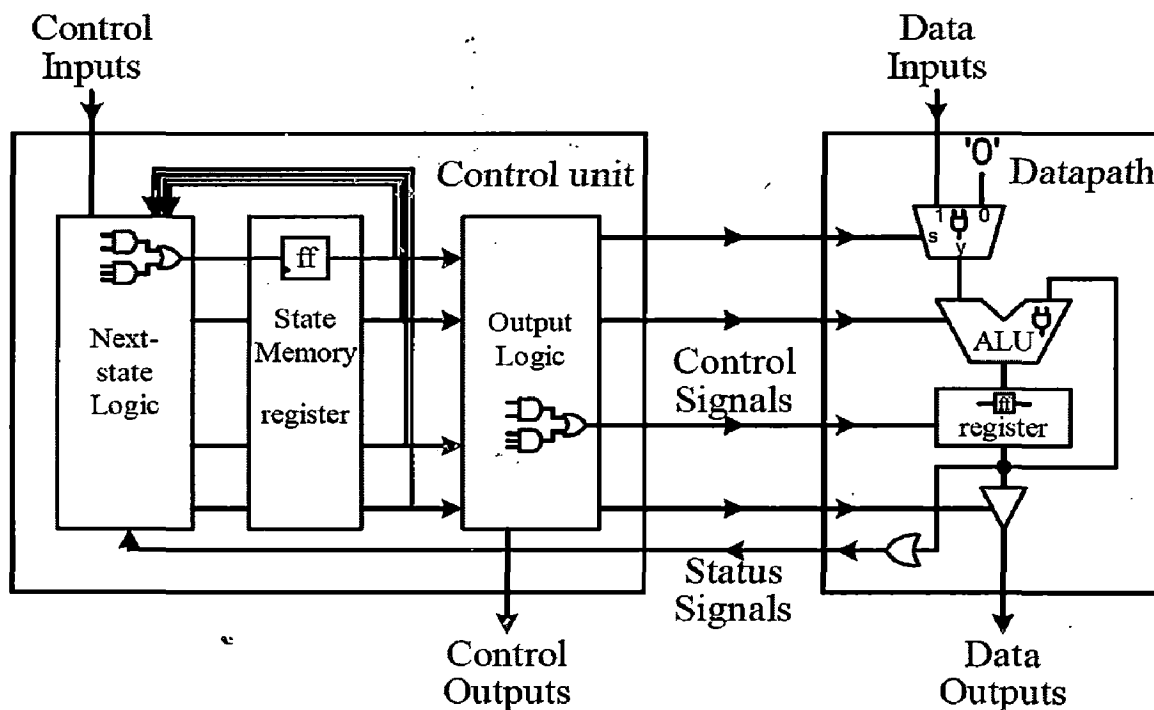


Figure 4.19. Block diagram of the Control Path

Depending on the current state, the control inputs and the status signals, the next-state logic in the control unit will determine what state to go to next in the next clock cycle. Thus, the control unit is also referred to as a finite-state machine (FSM) because of this. In every state, the output logic that is in the control unit generates all the appropriate control signals for controlling the datapath. The datapath, in return, provides status signals for the nextstate logic.

Architecture of the control unit contains two processes as shown in figure 4.20. The first is a combinational process (not clocked) that examines the current state and all inputs and produces output control values and next state output. The second is a Sequential process (has a clock) that is used to store the current state and copy the next state to the current state. The next state transitions occur on rising edges of the clock input. The control block is a very large state machine that contains a number of states for each instruction. Executing all of the states for an instruction performs the necessary steps to complete the instruction.

If the reset signal is high, the sequential process labeled `controlffproc` sets signal `current_state` to state value `reset1`. This is the first state of the reset sequence for the CPU. This state starts the process of getting the CPU ready to execute instructions.

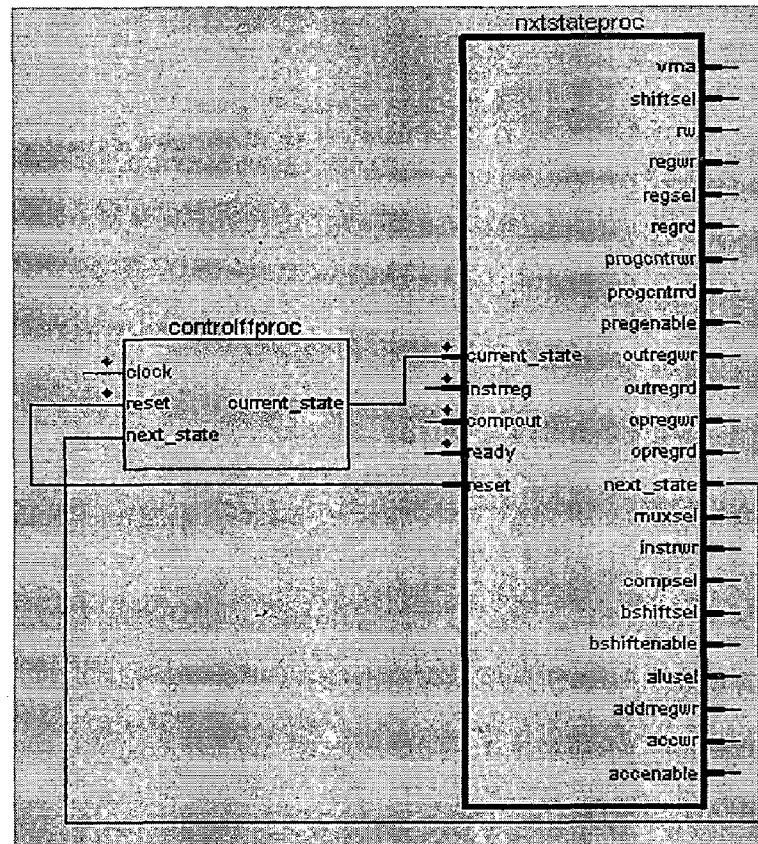


Figure 4.20. Controller Block diagram

If the reset signal is not '1' and there is a rising edge on the clock signal, then the `controlffproc` process copies the `next_state` signal generated by the combinational process to signal `current_state`. This is the method for the state machine to advance from one state to another.

After the reset signal is set to a value other than '1', the state machine is in state `reset1`. This state causes the alu entity to output the value 0, the shift entity to pass the value with no modification, and the next state signal to be updated with the value `reset2`.

At the next clock edge, the state machine advances to state `reset2`. State `reset2` leaves the control signals for the alu and shift entities as before, but also sets the `OutRegWr` signal to a '1', causing the 0 value on the data bus to be

written to register OutReg. The goal of the reset sequence is to set up the program counter to start reading instructions from memory.

After state reset2, the state machine next goes to state reset3 on the next clock edge. This state sets signal OutRegRd to a '1', causing entity OutReg to output its value to the data bus. The state machine then advances to state reset4. During reset4, the value from OutReg is copied into register ProgCntr and also to register AddrReg. The state machine advances to state reset5, sets output signal RW (read write) to '0' (read mode), and signals VMA (Valid Memory Address) to a '1'. This causes memory entity mem to output the data at location 0 to the data bus. The state machine advances to state reset6 and, depending on the value of the ready signal from the memory, either stays in reset6 or writes the memory data value to register InstrReg and goes to state execute.

At this point, the state machine has reset the state of the CPU to a known state and loaded the first instruction into register InstrReg. From this point forward, the state machine changes state depending on the instructions encountered.

4.5 CPU :

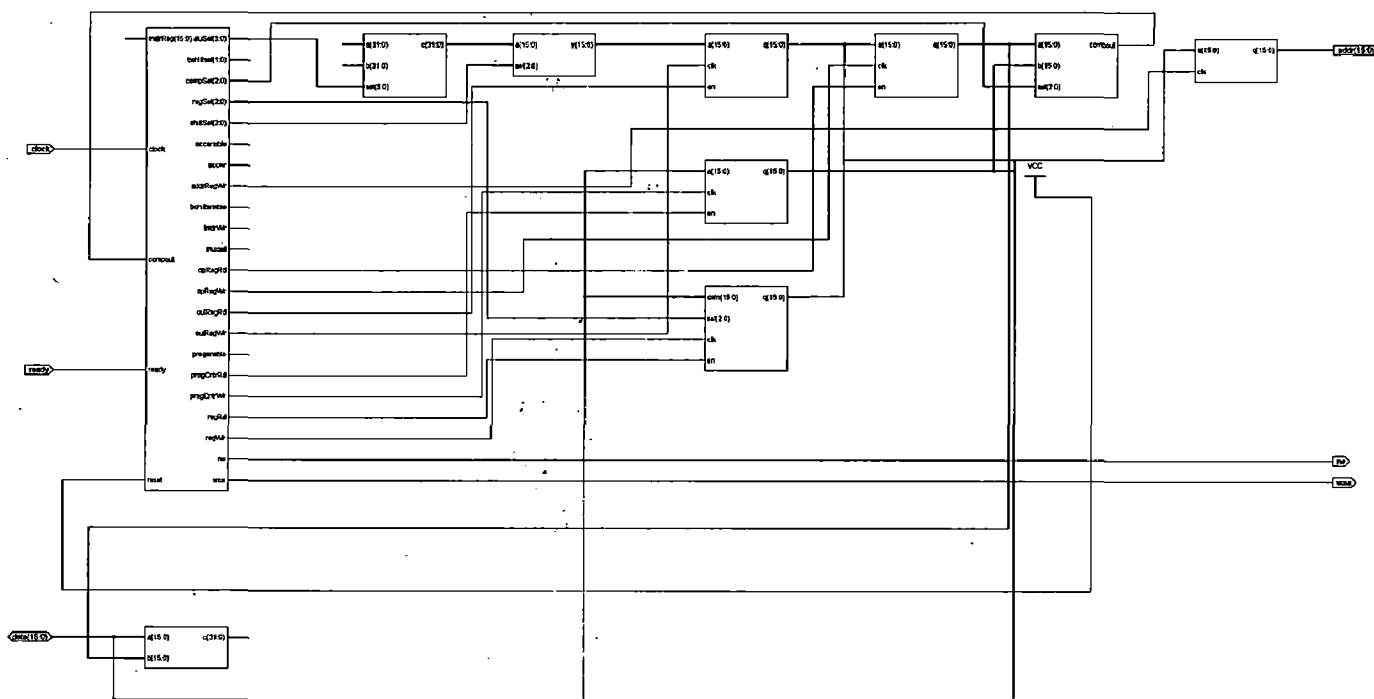


Figure 4.21 RTL Diagram of CPU

The figure 4.21 shows interconnection of the all the blocks to form the complete DSP Processor CPU core. All of these units communicate through a common, 16-bit tristate data bus. The processor fetches instructions from the memory and executes these instructions to run a program. These instructions are stored in the instruction register and decoded by the control unit. The control unit causes the appropriate signal interactions to make the processor unit execute the instruction.

When executing an instruction, a number of steps take place. The program counter holds the address in memory of the current instruction. After an instruction has finished execution, the program counter 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 program counter is loaded with the next instruction location directly.

The control unit copies the program counter value to the address register, which outputs the new address on the address bus. At the same time, the control unit sets the R/W (read write signal) to a '0' value for a read operation and sets signal VMA (Valid Memory Address) 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.

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

Chapter 5

Simulation Results

The processor module is designed by interconnecting the basic blocks, through the structural model in VHDL. All the entities of the processor were developed using behavioral model and structural model in VHDL. The simulation results of the each stage and the simulation result of the processor on a whole is shown in next few pages.

Figure 5.1 shows the simulation result of the ALU. The inputs are control signal **sel**, input data (**a** and **b**) and output is the result(**c**) from ALU. The control signal **sel** selects the type of the operation, and is performed.

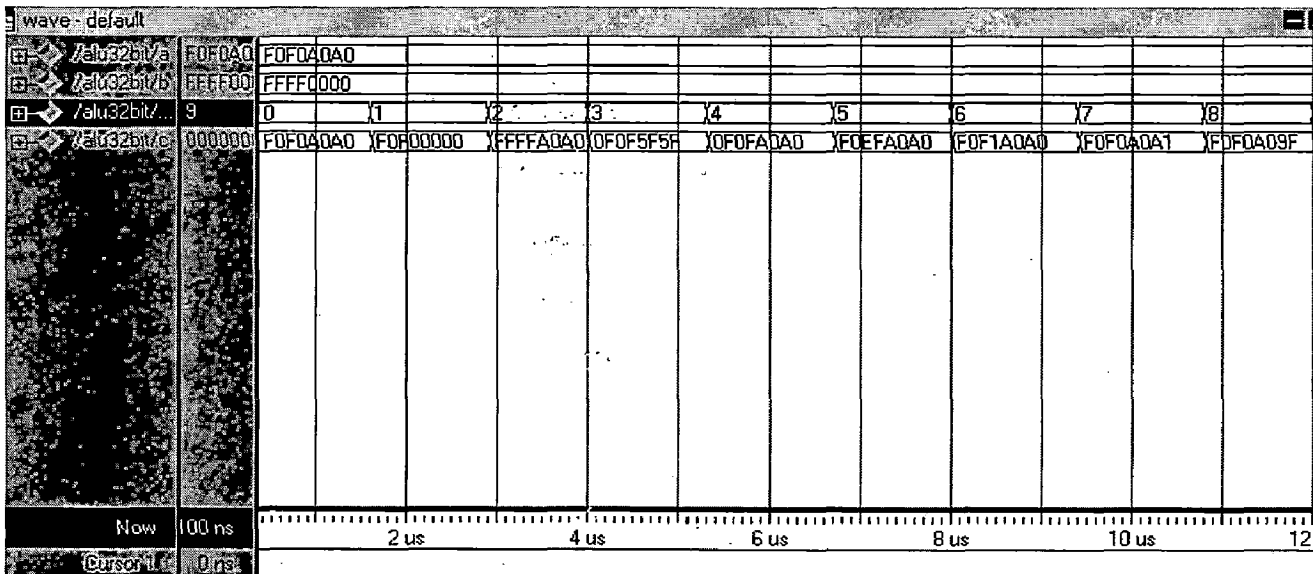


Figure 5.1 Simulation Result of ALU

Figure 5.2 shows the simulation result of B-Shifter. The input data is operated as per the control signals and produces the output.

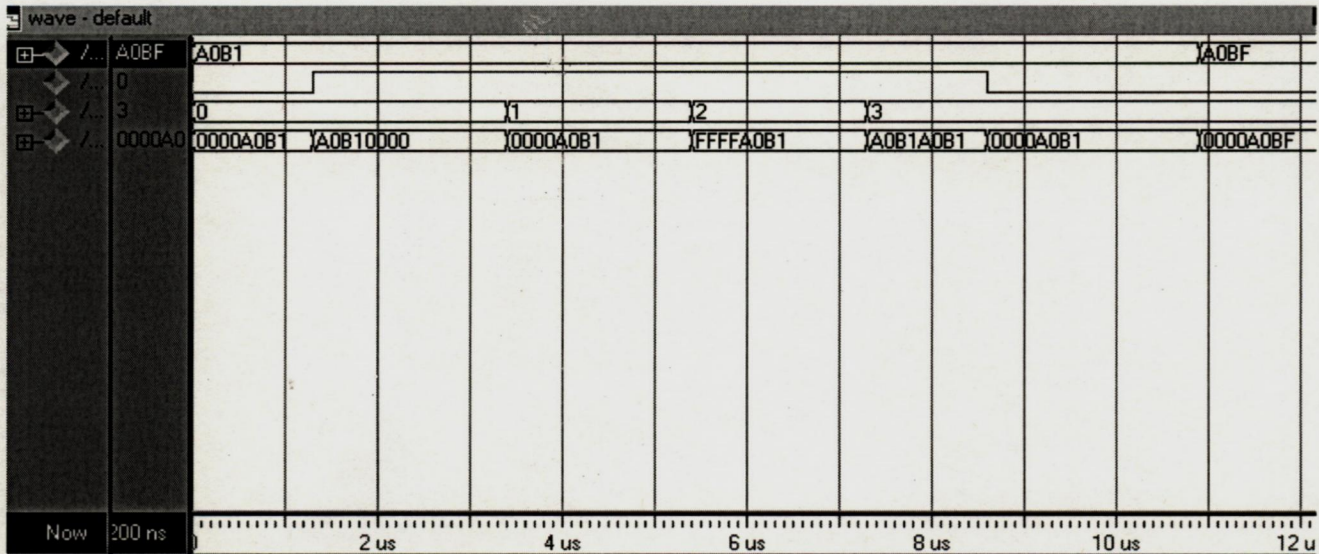


Figure 5.2 Simulation Result of B Shifter

Figure 5.3 shows the simulation result of tri register. 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. When a rising edge is applied to input **clk**, the data on input **a** is stored in the register. Input **en** is used to control output **q**. When **en** is a '1' value, the register state is driven to output **q**. When **en** is a '0', output **q** is a high impedance value.

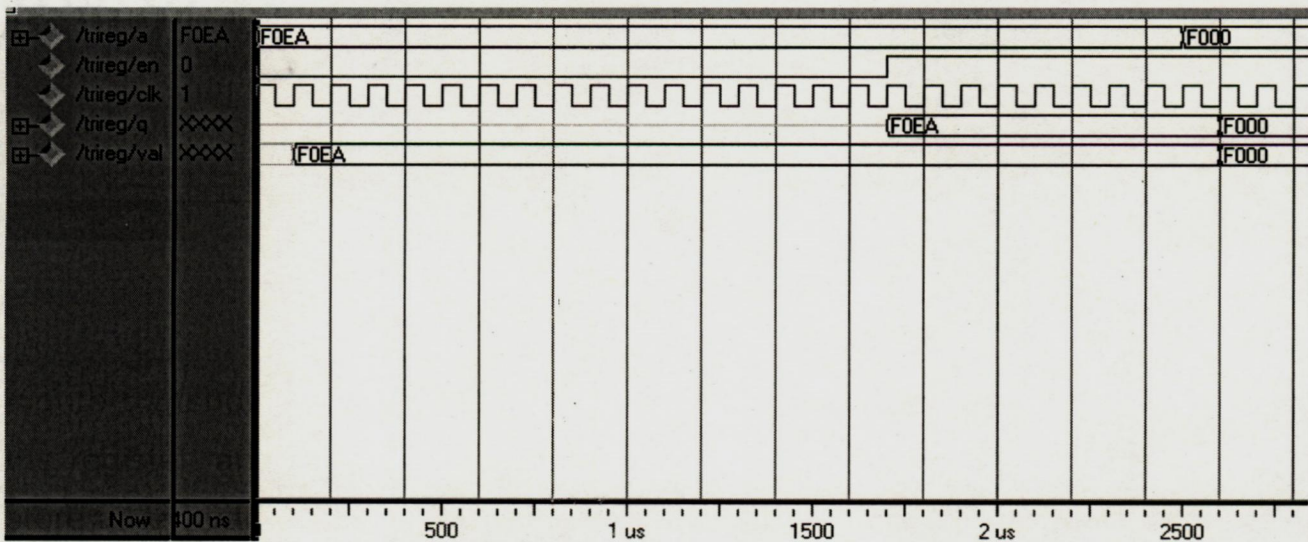


Figure 5.3 Simulation Result of Tri Register

Figure 5.4 shows the simulation result of Register Array. The input signal **sel** selects one of the registers and the input **data** is written into that register, whenever the input enable line **en** is high. From the simulation, at each time one of the registers got selected and modified.

Whenever input **sel** changes, the value of the signal **temp_data** gets updated. The signal **temp_data** is passed to the output if the **en** signal is '1'; otherwise, it puts out Z values. The Z values signify that the regarray entity is not driving the output when the **en** input is unasserted.

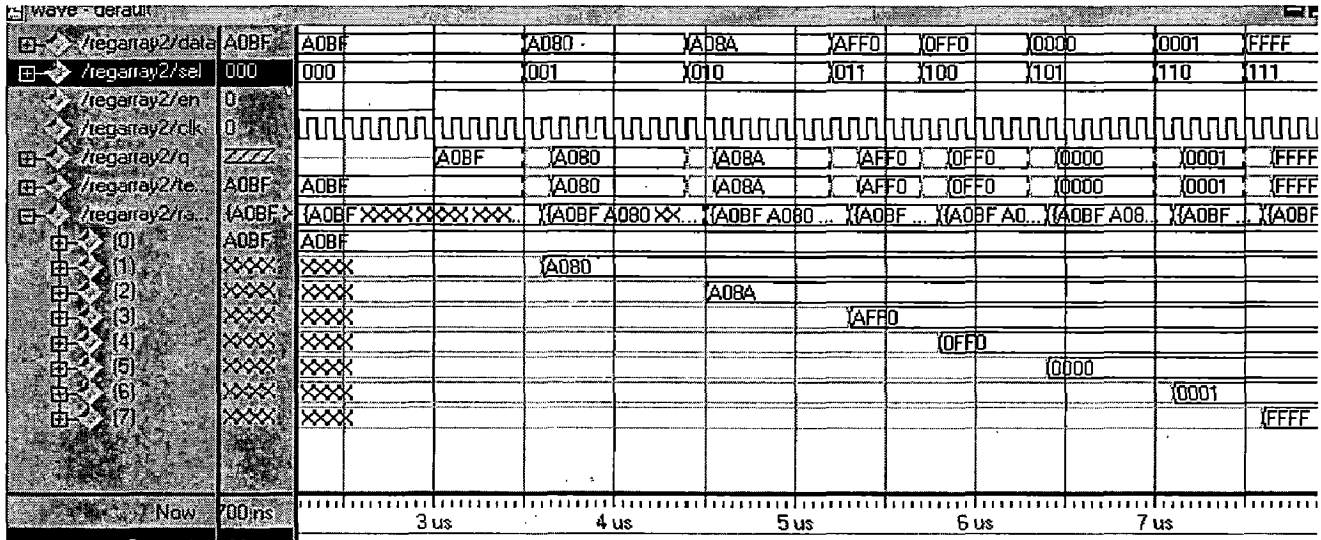


Figure 5.4 Simulation Result of Register Array

Figure 5.5 shows the simulation result of control unit whose inputs are the opcode part of the instruction and the outputs are different control signals. The control signals are applied to their corresponding components at the appropriate time.

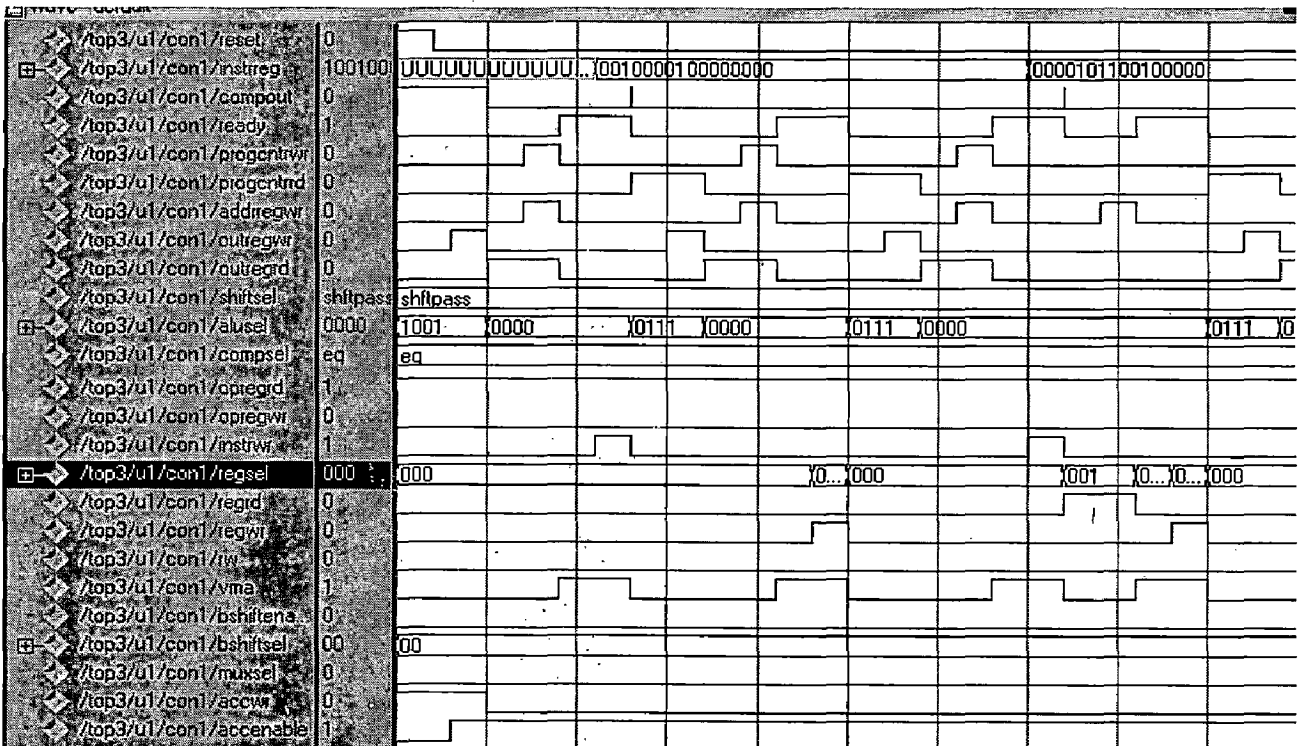


Figure 5.5 Simulation Result of Control Unit

Figure 5.6 shows the simulation result of 16-bit array Multiplier, where a, b are the inputs and c is the output. The inputs are 16 bit data and the output is of 32 bits. Remaining signals are the intermediate results. From the figure it is observed that the output is changed with the immediate change in the input(s), i.e. without any delay. The time to complete multiplication is less than 1 ns. The performance is improved than the multiplier of shift-add type.

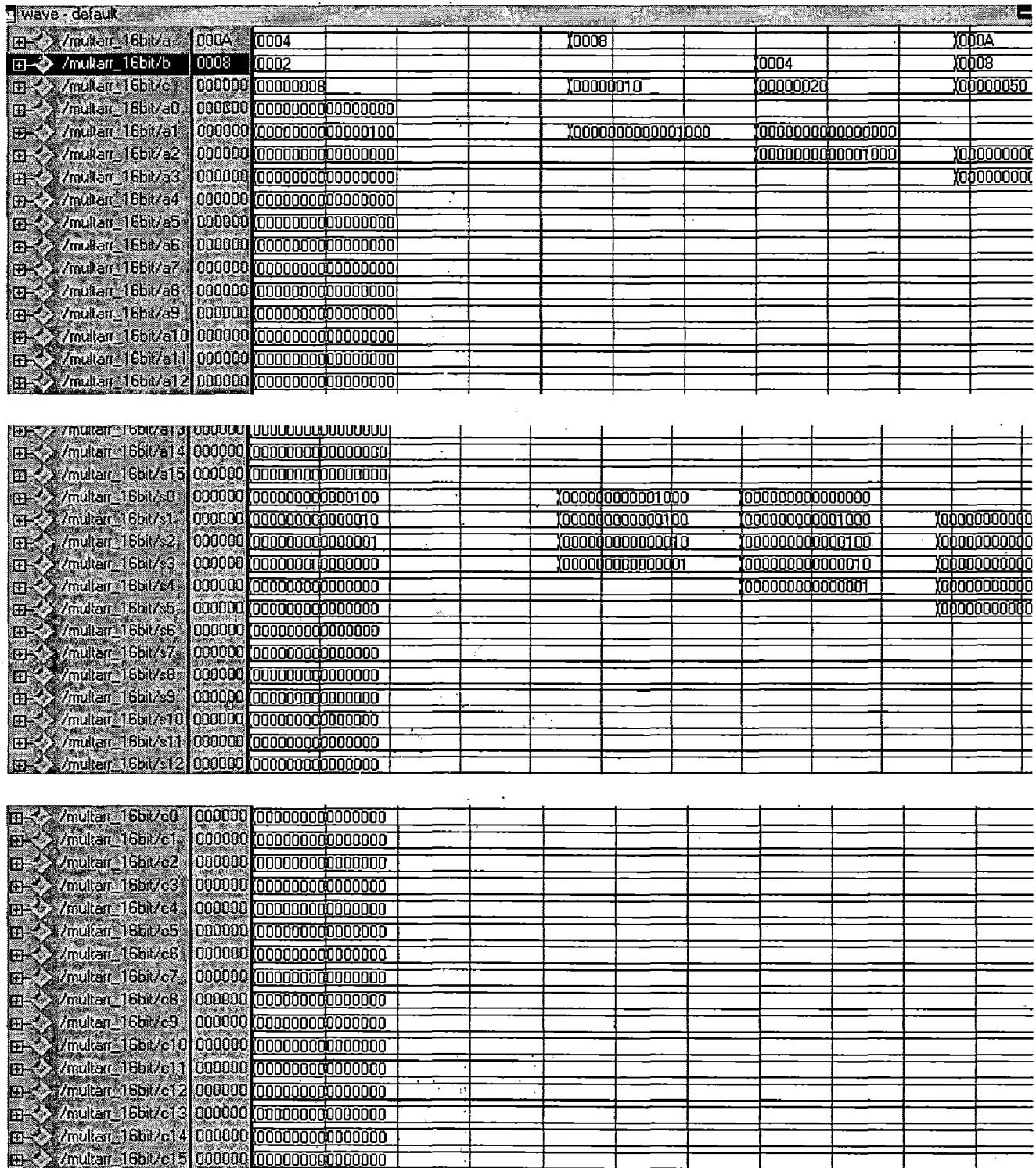


Figure 5.6 Simulation Result of 16 bit Array Multiplier

Figures 5.7 (a), (b), (c) show the simulation result of the main entity processor. The figures show the simulation result for a block of instructions that are executed. To cover some more instructions and their execution, these blocks of instructions are taken as the cases.

Case 1:

Instruction	Opcode
MVI R1,0010H	2100H 0010H
MOV R3,(R1)	0B20H
LDAI FFFFH	9000H FFFFH
ANA R1	8100H
INC R1	3900H

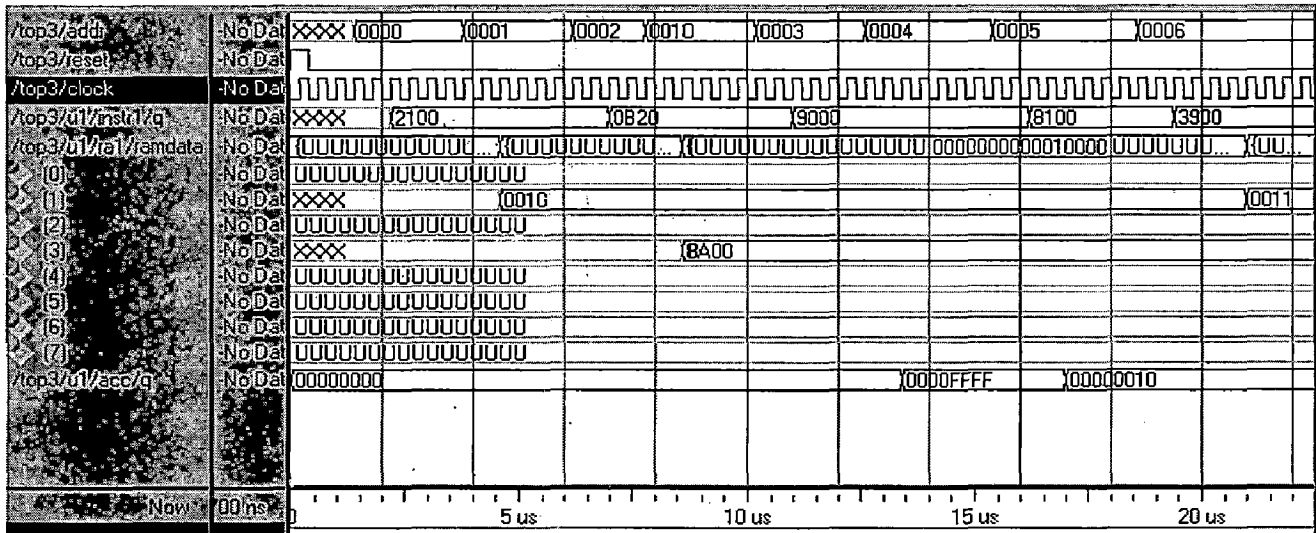


Figure 5.7(a) Simulation report of DSP Processor for case 1

Case 2:

Instruction	Opcode
MVI R1,0010H	2100H 0010H
MOV R3,(R1)	0B20H
STR R3, R1	1320H
MOV R2,(R3)	2100

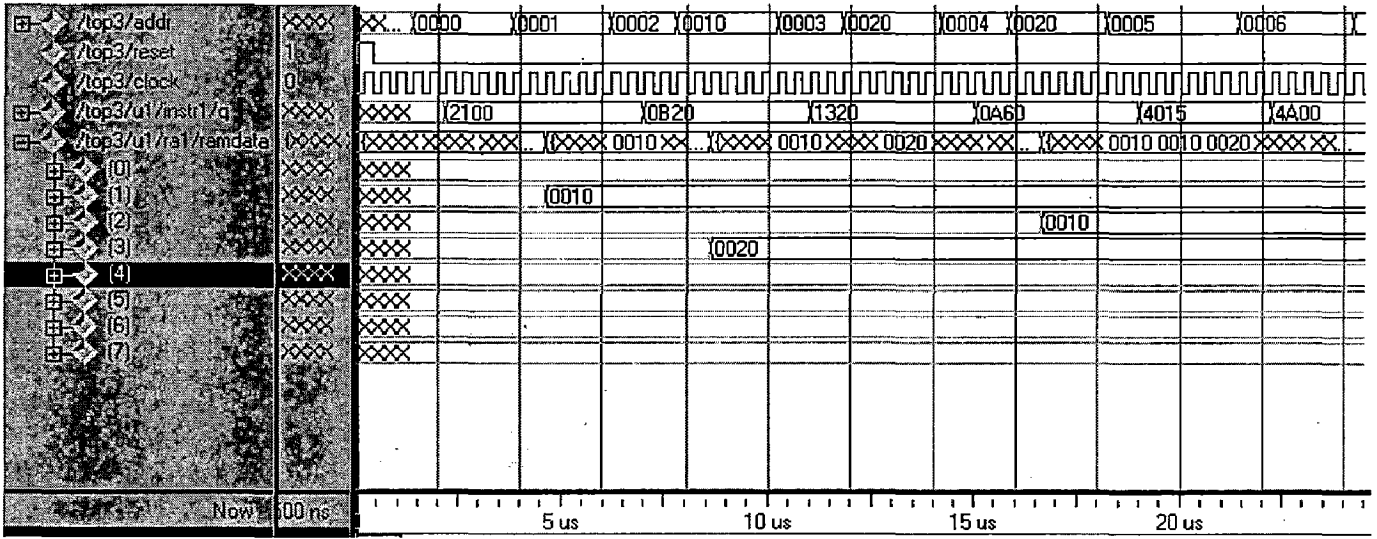


Figure 5.7 (b) Simulation report of DSP Processor for case 2

Case 3:

Instruction	Opcode
MVI R0,0004H	2000H 0004H
LT, 0015H	4015H
MPY R0	4800H
ANI 0010H	5000H 0010H
XRA R0	9800H
LDAI FFFFH	9000H FFFFH

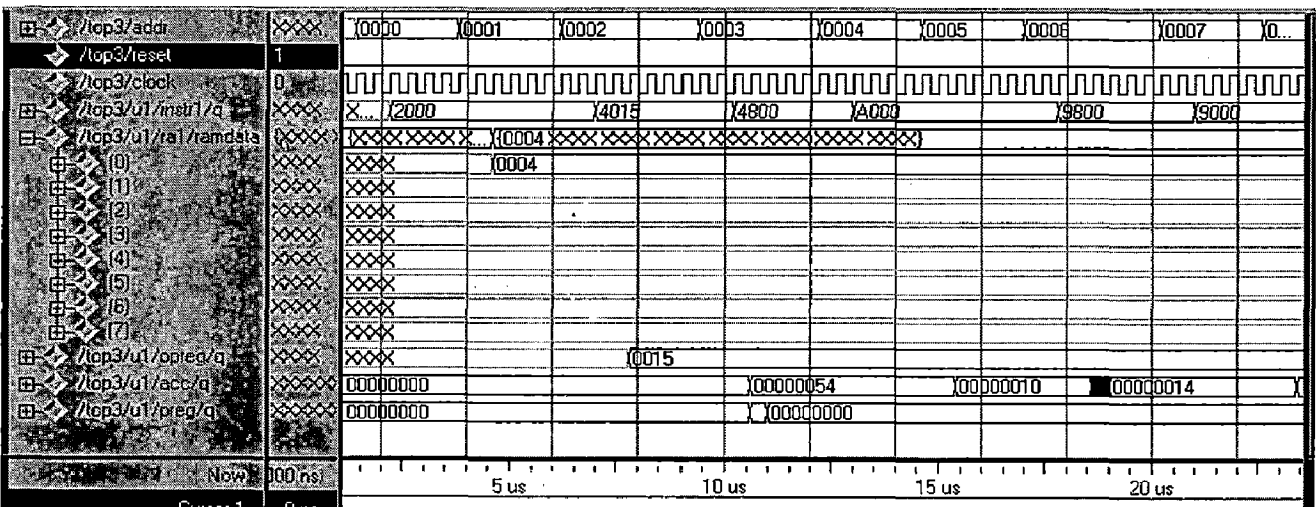


Figure 5.7 (c) Simulation report of DSP Processor for case 3

Case 4:

Instruction	Opcode
MVI R2,0004H	2200H 0004H
LDAI AAAAH	9000H AAAAH
JMP 0010H	2800H 0010H

The instruction OR R2, whose opcode is 8A00H, is there at the address location 0010H. That instruction is executed as shown in the result.

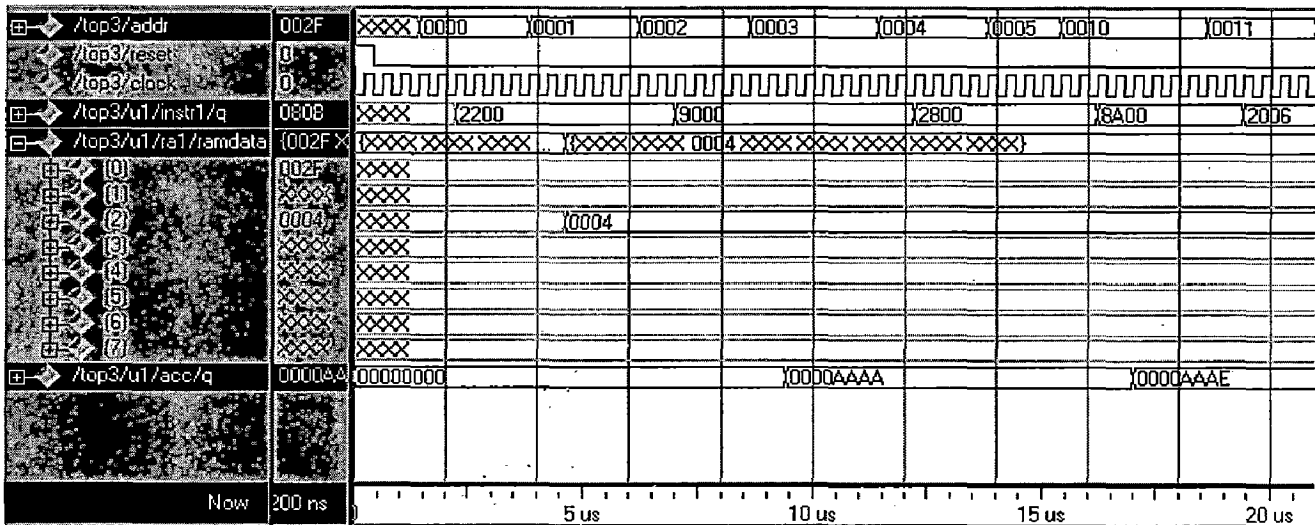


Figure 5.7 (d) Simulation report of DSP Processor for case 4

Chapter 6

Conclusion and Future Scope

6.1 Conclusion:

The present work deals with implementing a 16-bit DSP Processor on FPGA. The instruction set of the proposed processor is of 16 bit instructions and supports the numeric intensive signal processing operations and general purpose applications. The operands of the instructions are of 16 bits. The instruction set consists of primarily of single word instructions, only Immediate Data Type instructions are of two word instructions.

Each block of the processor is designed by behavioral model and structural model and these blocks are connected by structural model using VHDL. The simulation results have been provided in the previous chapter. The synthesized RTL diagrams of the each block is shown in the Appendix B.

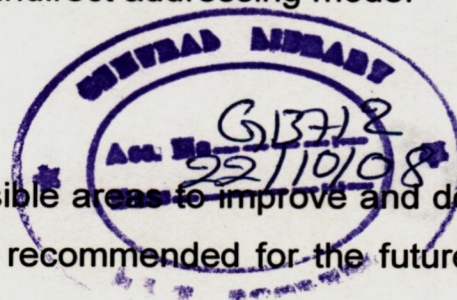
The 16 bit array multiplier is used in this present work. The array multiplier requires a large amount of logic, but can compute a product much more quickly than the method of shifting and adding typical of earlier computers.

The registers in the register array are used for temporary storage of data and are used to store the address of the operand in indirect addressing mode.

6.2 Scope For Future Research:

In terms of the future work, there are many possible areas to improve and do further development. The improvements which are recommended for the future are as follows

- Development of processor having advanced features like instruction look-ahead and prediction functionality.
- The current controller has specific states for each and every instruction, but could be improved by creating generic multi-cycle stages for the execution of each instruction.



- Researching new and improved instructions to increase the usability of the DSP.
- By using pipelining the long paths can be reduced and the speed of the processor can be improved.
- The instruction set can be extended including parallel MAC and distributed arithmetic instructions.
- An attempt can be made to extend the work for floating point numbers.

References

- [1]. Chin-Long Wey And Jin-Fu Li , "Design Of Reconfigurable Array Multipliers And Multiplier-Accumulators," The 2004 IEEEAsia-Pacific Conference On Circuits And Systems, pp 37-40, December 6-9, 2004.
- [2]. Chang Choo, Jeff Chung, James Fong, Shinghin Eddy Cheung, "Implementation of Texas Instruments TMS32010 DSP Processor on Altera FPGA", San Jose State University, 2004.
- [3]. TMS320C1x Digital Signal Processors", <http://focus.ti.com/lit/ds/symlink/smj320c15.pdf>, pp. 8-10,13, 1991.
- [4]. Guoping Wang; Shield, J, "The efficient implementation of an array multiplier", Electro Information Technology, May 2005.
- [5]. U.Meyer-Bease, "Digital Signal Processing With Field Programmable Gate Arrays," Springer Publications.
- [6]. Wayne Wolf, "FPGA-Based System Design," Prentice-Hall, 2004.
- [7]. M. Morris Mano, "Digital Design," Second Edition, 2001.
- [8]. Aamir A. Farooqui, Vojin G. Oklobdzija, "General Data-Path Organization of a MAC unit for VLSI DSP Processors," IEEE 1998.
- [9]. Charles H. Roth, Jr, "Digital System Design Using VHDL," Fifth Edition, 2004.
- [10]. Douglas L Perry, "VHDL Programming by Example," Fourth Edition.
- [11]. J Bhaskar, "A VHDL Primer", third edition, Pearson Prentice Hall.
- [12]. Surin Kittitornkun and Yu Hen Hu, "Programmable Digital Signal Processor(PDSP): A Survey"; University of Wisconsin in Madison, 2003.

- [13]. William Stallings, "computer organization & architecture", sixth edition, 2003.
- [14]. Ney Laert Vilar Calazans, Fernando Gehm Moraes² César Augusto Missio Marcon, "Teaching Computer Organization and Architecture with Hands-On Experience", 2002 IEEE 32nd ASEE/IEEE Frontiers in Education Conference, November 6-9, 2002, Boston, MA.
- [15]. Texas Instruments, Inc., "TMS 320C1x User's Guide", 2563968-9721 revision, July 1991.
- [16]. Sen M Kuo, Bob H Lee, Wenshun Tian, "Real Time Digital Signal Processing Implementations and Applications" Second Edition, 2006.
- [17]. Hassan M. Ahmed, Richard B. Kline, "Recent Advances in DSP Systems", IEEE Communications Magazine, May 1991.
- [18]. John G. Proakis, Dimitris G. Manolakis, "Digital Signal Processing", third edition.
- [19]. Jonathan and Michelle, "Micro Processor Design Principles and Practices with VHDL ", 2004.
- [20]. Volnei A. Pedroni, "Circuit Design with VHDL", Massachusetts institute of technology, 2004.
- [21]. Peter J. Ashenden, "The VHDL Cookbook", First Edition, July, 1990.
- [22]. Vijay K. Madisetti and Douglas B. Williams, "Digital Signal Processing Hand Book", Georgia Institute of Technology, 1991.
- [23]. Peter J. Ashenden, "The VHDL Cookbook", First Edition, 1990.
- [24]. <http://www.xilinx.com/support/library.htm>
- [25]. www.mte-india.com
- [26]. www.wikipedia.com

Appendix A

Software Details

In the present work, there were many softwares which were utilized. Some of these were utilized for the simulation purpose, some of them were used for verification purpose and some for the validation purpose. The hardware here means the FPGA development kit, on which the developed design is implemented. The Details are discussed in the following sections.

A.1 Softwares used:

1. Hardware Descriptive Language (HDL)
2. Xilinx ISE 7.1

A.1.1 Hardware Descriptive Language:

A Hardware Descriptive Language (HDL) is a computer language designed for formal description of electronic circuits. It can describe a circuit operation, its structure, and the input stimuli to verify the operation (using simulation). A HDL model is a text-based description of the temporal behaviour and / or the structure of an electronic system. In contrast to a software programming language, the HDL syntax and semantics include explicit notations for expressing time and concurrent execution, which are the primary attributes of hardware.

Traditional programming languages such as C/C++ (augmented with special constructions or class libraries) are sometimes used for describing electronic circuits. They do not include any capability for expressing time explicitly and, consequently, are not proper hardware description languages.

There are over 200 Hardware Descriptive Languages (HDLs) around the world. Only two HDLs which are famous are VHDL and Verilog. VHDL stands for HSIC Hard-ware Descriptive Language. Both VHDL and Verilog are powerful languages that allow describing and simulating complex digital systems. Verilog

is popular in Silicon Valley companies, while VHDL is used more by governments, in Europe, Japan and most of the universities worldwide. Major Computer Aided Design (CAD) frameworks now support both languages.

Features of VHDL:

There are many properties of VHDL which make it very convenient for Hardware Design Engineers. These are concurrent signal assignment, process execution, statement concurrency, operator overloading, packages, component instantiation. The difference which is clearly visible among C programming and VHDL is that, statements in C language are executed sequentially whereas in VHDL, statements can be executed sequentially or concurrently or one can have both, i.e., some of the statements being executed in sequential and some in concurrent fashion.

A.1.2 Xilinx ISE 7.1

Xilinx ISE stands for Xilinx Integrated System Environment (ISE). ISE controls all aspects of the design flow. Through the Project Navigator interface, one can access all of the design entry and design implementation tools. One can also access the files and documents associated with the project. Project Navigator maintains a flat directory structure; therefore, the project should be updated through the use of snapshots.

The Xilinx ISE system is an integrated design environment that consists of a set of programs to create (capture), simulate and implement digital designs in a FPGA or CPLD target device. All the tools use a graphical user interface (GUI) that allows all programs to be executed from toolbars, menus or icons.

Design Entry:

The first step is to enter the design. This can be done by creating "Source" files based on the design criteria. Source files can be created in different formats such as a schematic, or a Hardware Description Language (HDL) such as VHDL, Verilog. A project design will consist of a top-level source file and various lower-

level source files. Any of these files can be either a schematic or a HDL file. Source files are used to describe hardware for the purpose of simulation, modeling, testing, design and documentation of digital systems.

Simulation

This is an important step that should be done at various stages of the design. The simulator is used to verify the functionality of a design (functional simulation), the behavior and the timing (timing simulation) of the circuit. The operation of the design is verified before the implementation it as hardware. Simulation can be done using ModelSim in ISE software. ISE tool supports the following simulation tools:

- HDL Bencher is an automated test bench creation tool. It is fully integrated with project navigator.
- ModelSim from Model Technology is integrated in Project Navigator to simulate the design at all the steps.

Design Constraints

Setting constraints is an important step in the design process. It allows to control timing optimization and enables more efficient use of synthesis tools and implementation processes. This efficiency helps minimize runtime and achieve the requirements of design. There are many different types of constraints that can be set.

Design Synthesis

After the design has been successfully analyzed, the next step is to translate the design into gates and optimize it for the target architecture This is the synthesis phase. In this step the design is translated into gates and it will be optimized for the target architecture. The synthesis step creates netlist files, which is a description of how the circuit is realized or connected using basic gates, from the various source files. The netlist files can serve as input to the implementation module.

Design Implementation

After generating the netlist file (synthesis step), the implementation will convert the logic design into a physical file that can be downloaded on the target device i.e. FPGA. This step involves three sub-steps: Translating the netlist, Mapping and Place & Route.

This builds a physical realization of the synthesized logic by placing the logic blocks into logic elements and choosing interconnect resources for the connections between them. This creates the physical realization.

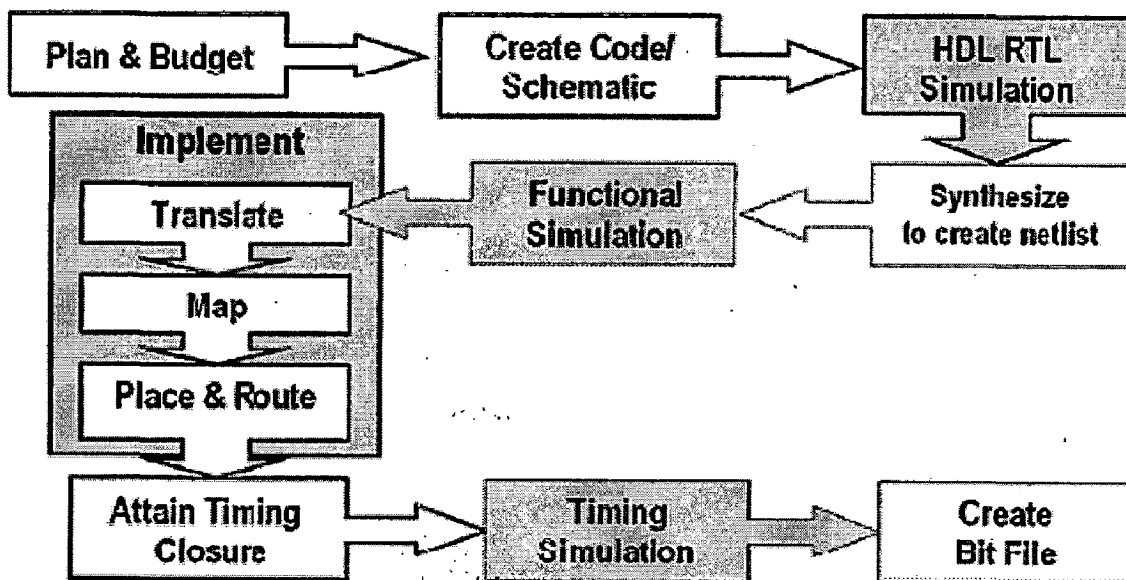


Figure A.1: Overview of various steps involved in the design flow

Generate bit stream

It creates the information needed to program the device. This information is called as bit stream file.

Device Configuration

This refers to the actual programming of the target FPGA by downloading the programming file, to the FPGA or any other FPGA development kit. It is the process by which the bit stream of a design, as generated by Xilinx software, is loaded into the internal configuration memory of the FPGA. SPARTAN-2 device supports both serial configuration, using master/slave serial and JTAG modes, as

well as byte-wide configuration employing the select Map mode. FPGAs support 3 configuration modes.

- Slave Serial Mode
- Master Serial Mode(Through PROM)
- Boundary Scan Mode

The configuration mode is determined by the three configuration mode pins (M1, M2, M3).

Sources Window

This window contains the design source files for a project. These are the source files that are created or added to the project. A drop down list at the top of sources window allows you to select source files that are associated with a particular design aspect such as Synthesis/Implementation or Simulation.

Processes Window

The processes windows list the available processes (corresponding to the process selected in the processes window). Typically a particular process that is desired to be performed on the selected source file, is selected. This can include a simulation, implementation, etc. To run a process you can double click on the process. When a process has been successfully executed a green tick-on icon appears. When a high-level process is run, the Project Navigator will automatically run all the associated lower-level processes. An overview of the digital design flow is explained in Figure. A.1

Appendix B

Hardware details

B.1 Universal DSP Trainer

Universal DSP protoboard supports various FPGAs and is useful to physically verify DSP algorithms or simple designs.

B.1.1 Advantages of using FPGA

Using FPGA's for implementing DSP functions is one of the preferred ways, and has many advantages over using DSP processors and ASIC's.

- High performance
- Reconfigurable
- One chip solution

B.1.2. Spartan-2 :

Spartan-2 family is second generation high volume production FPGA solution. Devices in this family are available up to 200k gates, with up to 200 MHz system performance.

Spartan-II FPGA Family Members

Device	Logic Cells	System Gates (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O ⁽¹⁾	Total Distributed RAM Bits	Total Block RAM Bits
XC2S15	492	15,000	8 x 12	96	86	6,144	16K
XC2S30	972	30,000	12 x 18	216	92	13,824	24K
XC2S50	1,728	50,000	16 x 24	384	176	24,576	32K
XC2S100	2,700	100,000	20 x 30	600	176	38,400	40K
XC2S150	3,888	150,000	24 x 36	864	260	55,296	48K
XC2S200	5,292	200,000	28 x 42	1,176	284	75,264	56K

B.1.2.1. Features of Spartan 2 families are:

- On chip RAM
- Dedicated carry logic for high speed arithmetic

- Dedicated multiplier support
- Low power segmented routine architecture
- 16 high performance interface standards
- 4 dedicated delay locked loop(DLL) for advanced clock control
- Unlimited reprogram ability
- Very low cost

B.2 DSP Protoboard features:

- It supports a variety of FPGA's in different sizes and architectures
- Analog interface
 - Analog input (0-5 v) : 4channels using ADC
 - Analog output (0-5v) : 4 channels using DAC
- Function generator(using IC 8038)
 - With sine, square and triangular waveforms.
 - Frequency variable from 60-200hz.
- One anti aliasing filter at the input of ADC
- One reconstruction filter at the out put of DAC
- User interface
 - 16 output LEDs
 - 16 DIP switches
 - 4 key switches
 - 4 seven segmented displays
- Serial interface :One RS-232 channel using MAX3223
- User selectable configuration modes
- On board RPS

B.2.1 Power Supply:

FPGA supplies (V_{ccint} , V_{ccio}) are generated on the board

- V_{ccint} (Internal core supply voltage)=2.5v
- V_{ccio} (I/O bank supply voltage)=3.3v
- Download cable voltage=3.3v

B.2.2 LEDs

There are total 34 LEDs on the protoboard, which are grouped as

- POWER-ON LED
- DONE LED
- IL0 to IL15
- OL0 to OL15

B.2.3 DIP SWITCH

- 8-way DIP switches(SW1 & SW2): Used to apply logic inputs to FPGA
- SW7 is 4-way dip switch and is used to select different time constants for anti-aliasing filter
- SW9 is 4-way DIP switch and is used to select different time-constants for reconstruction filter
- SW8 is 4-way DIP switch and is used to select the frequency range of the function generator

B.2.4 Function Generator:

Function generator-IC8038 is used on board to generate sine, square, triangular waves in the frequency range of 60 Hz to 200 KHz. Output of the function generator can be used as the analog input to ADC for performing different DSP applications.

- Amplitude setting: The amplitude of generated waveform can be adjusted using potentiometers as follows:
 - PR7 for Sine wave
 - PR8 for Square wave
 - PR9 for Triangular wave
- Frequency setting: The frequency of the function generator can be varied in 2 steps
 - Coarse frequency – Using switch SW8
 - Fine frequency – Using potentiometer PR10

Appendix C

RTL Diagrams

This chapter gives the synthesized RTL diagrams of the blocks that are generated in the implementation of the processor. The RTL diagrams are obtained from Xilinx ISE 7.1 synthesis tool.

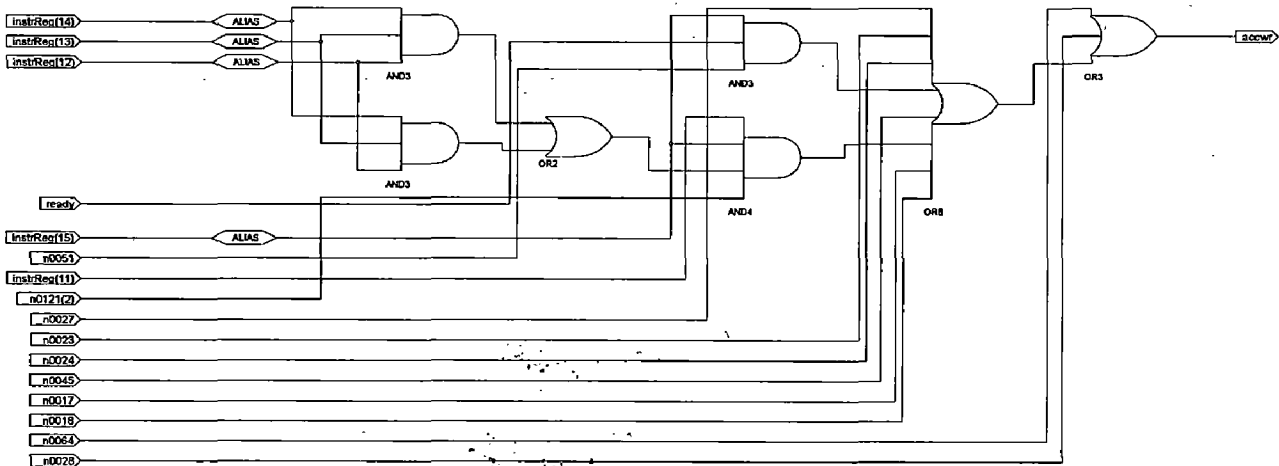


Figure B.1 RTL diagram for Accwr

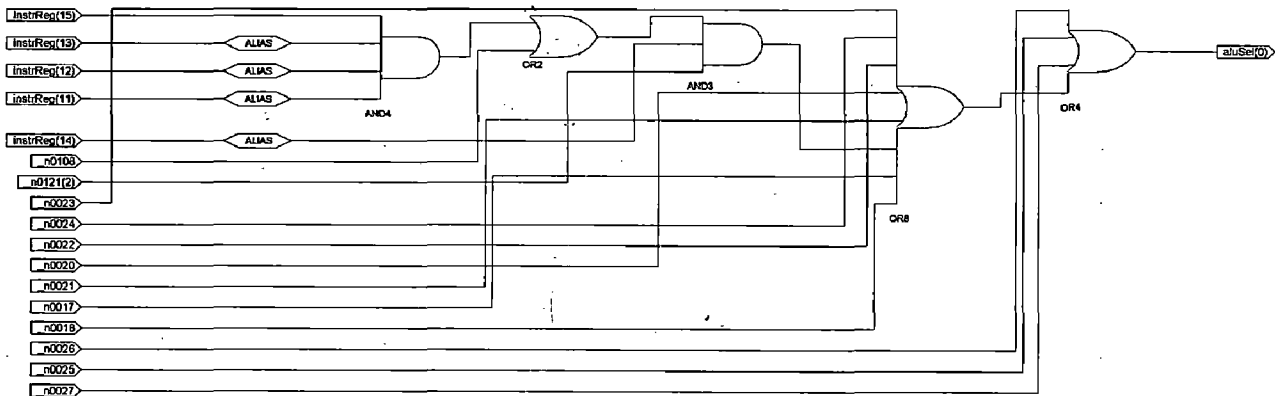


Figure B.2(a) RTL diagram for alusel(0)

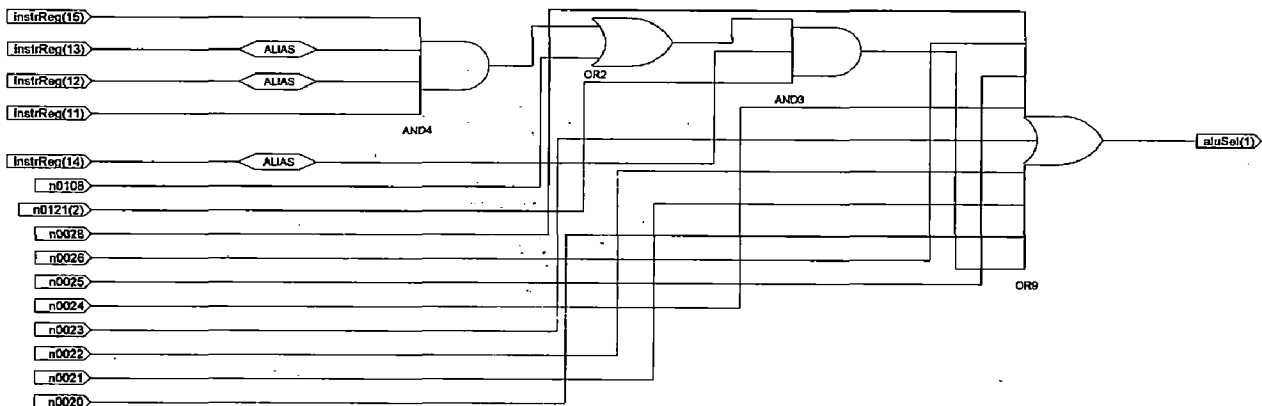


Figure B.2(b) RTL diagram for alusel(1)

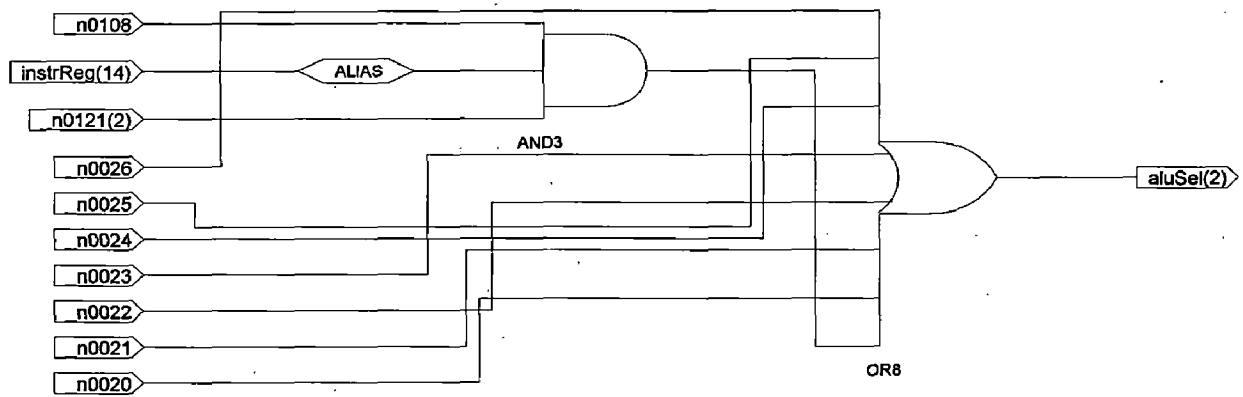


Figure B.2(c) RTL diagram for `alusel(2)`

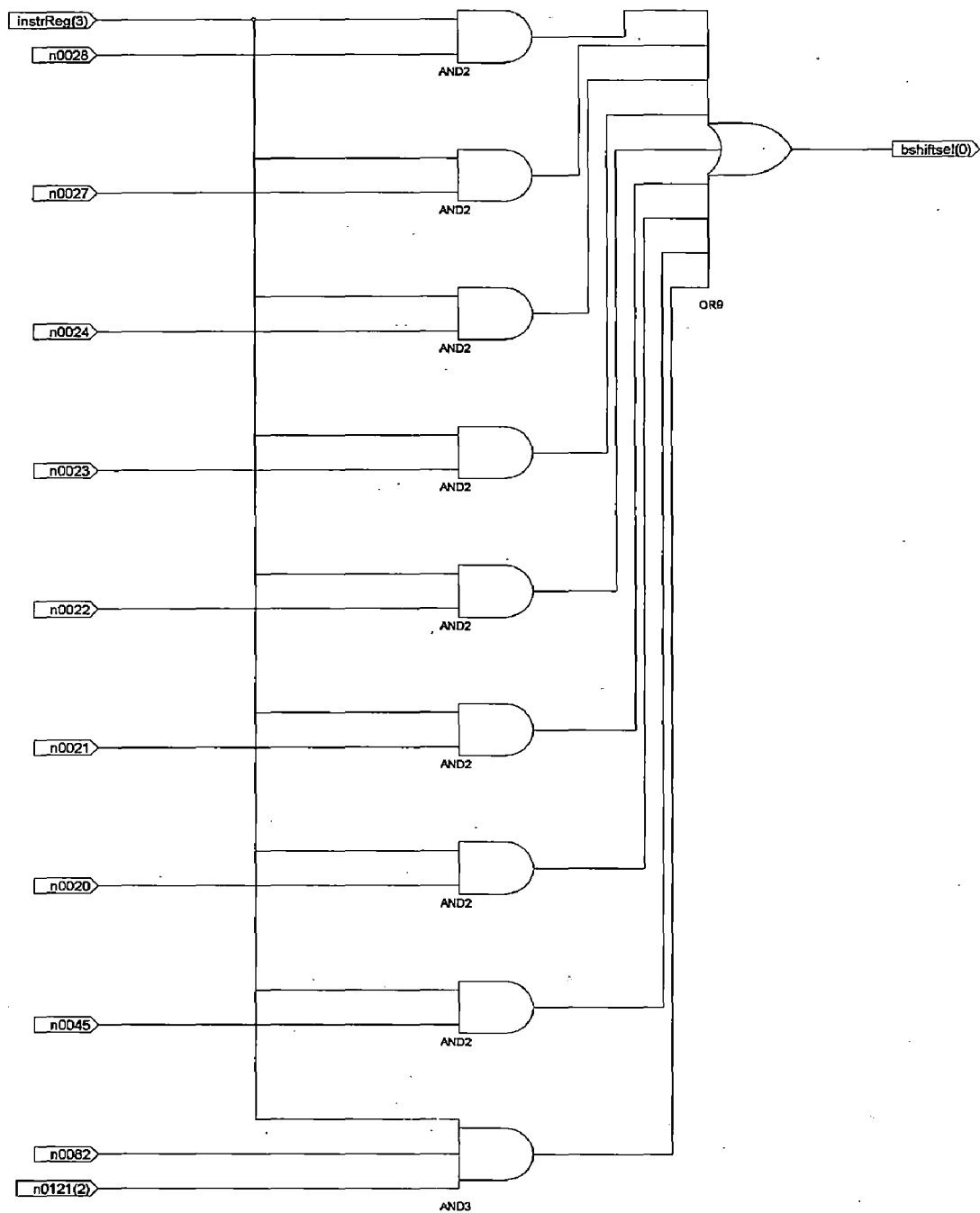


Figure B.3(a) RTL diagram for `bshftsel(0)`

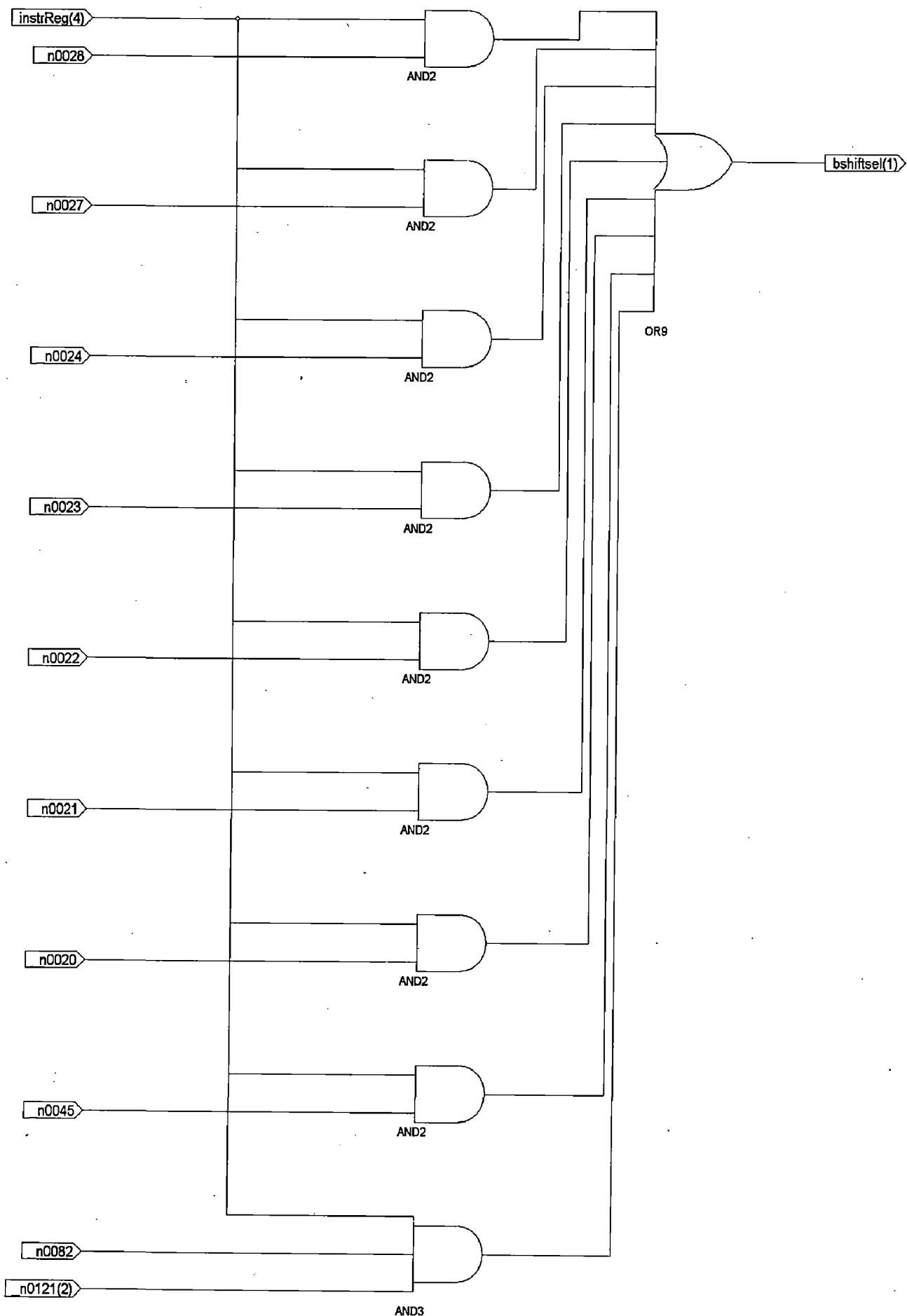


Figure B.3(b) RTL diagram for `bshftsel(1)`

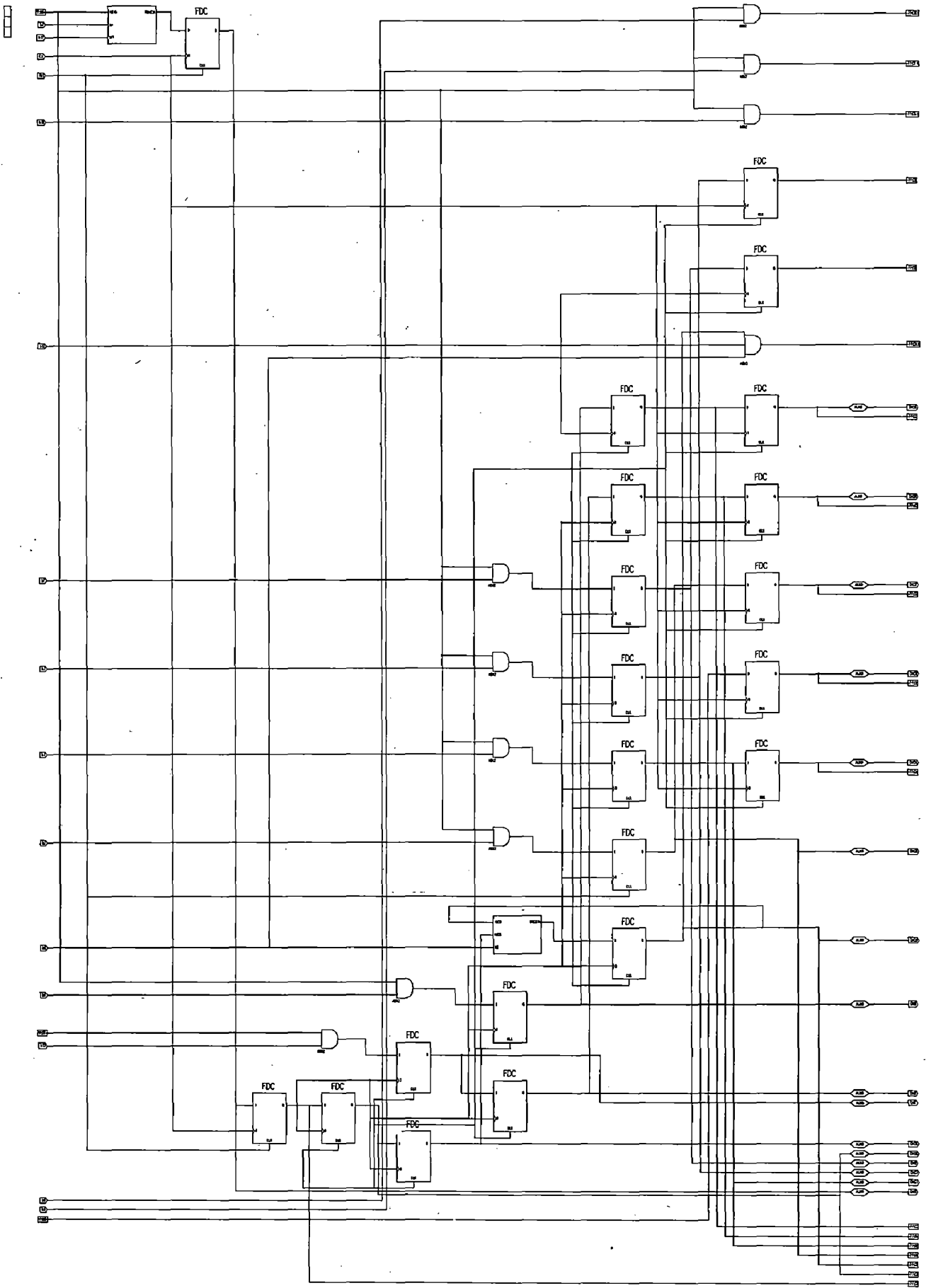


Figure B.4(a) RTL diagram for currentstate1

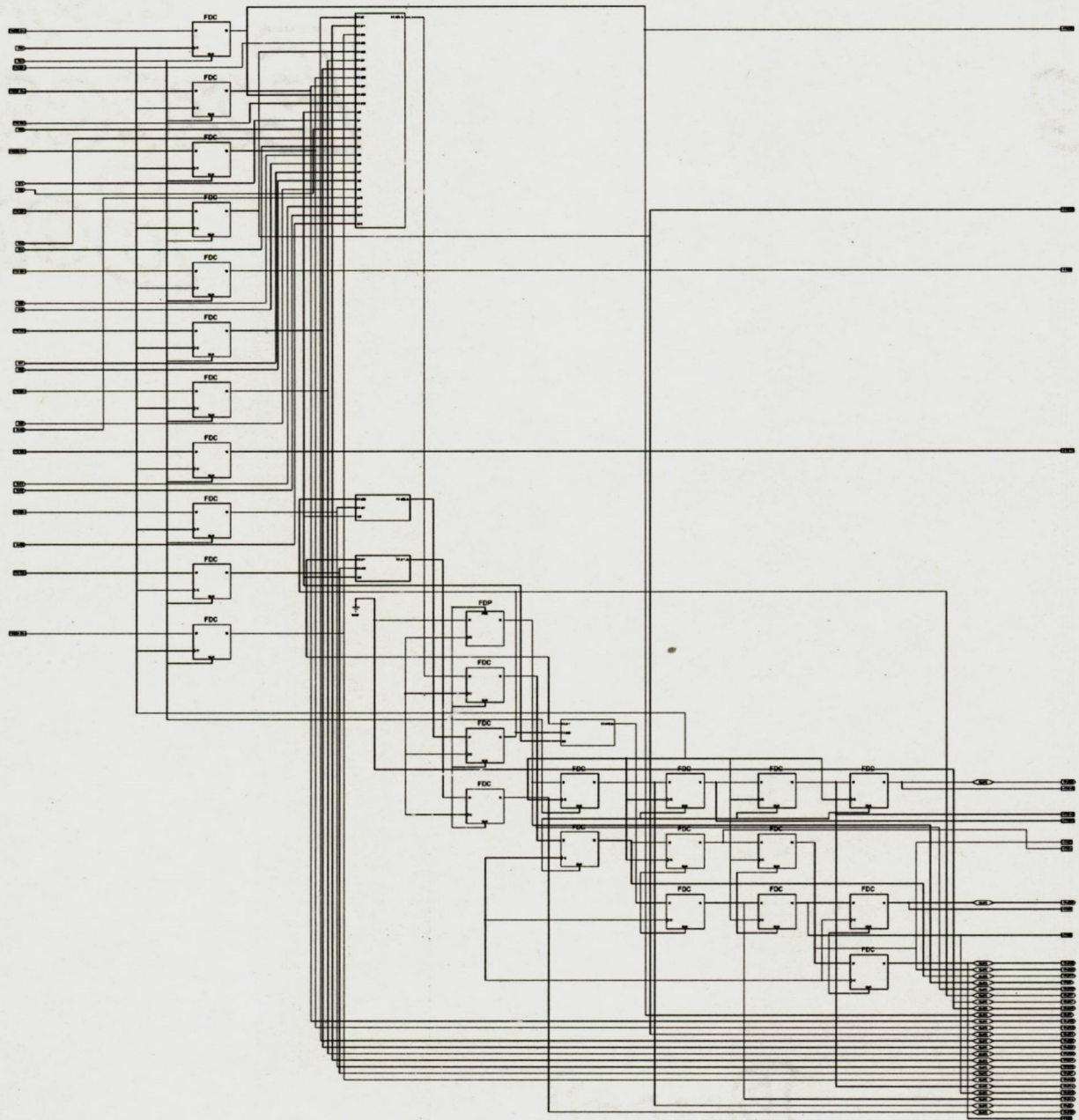


Figure B.4(b) RTL diagram for currentstate2

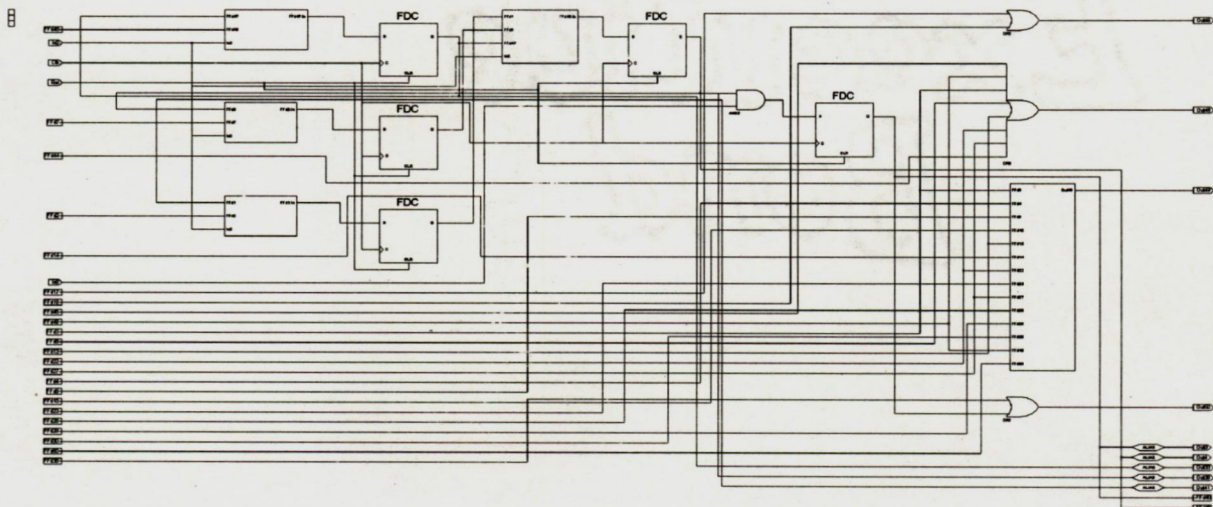


Figure B.4(c) RTL diagram for currentstate3

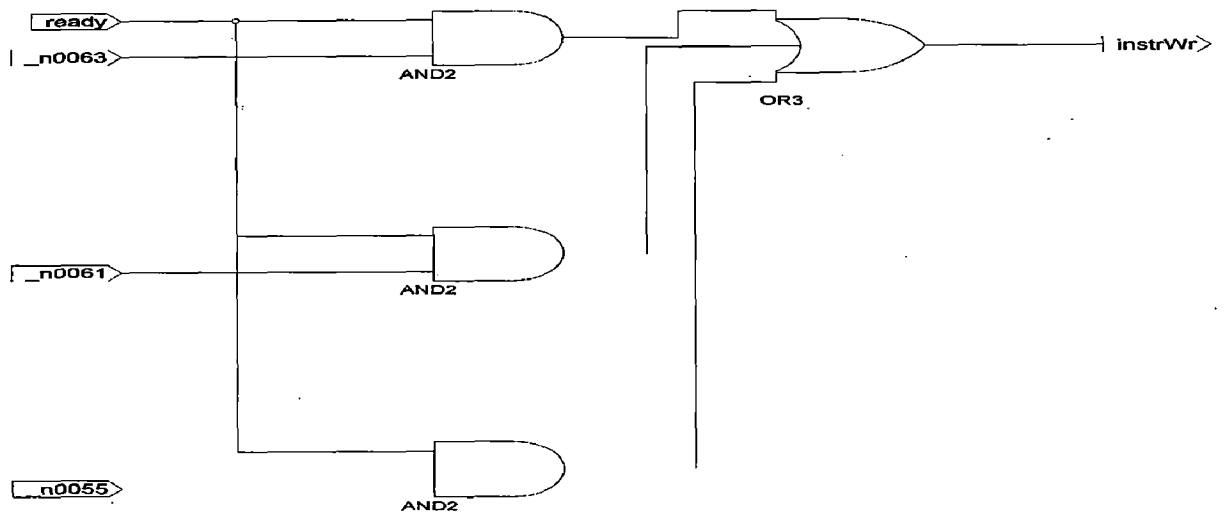


Figure B.5 RTL diagram for instrwr

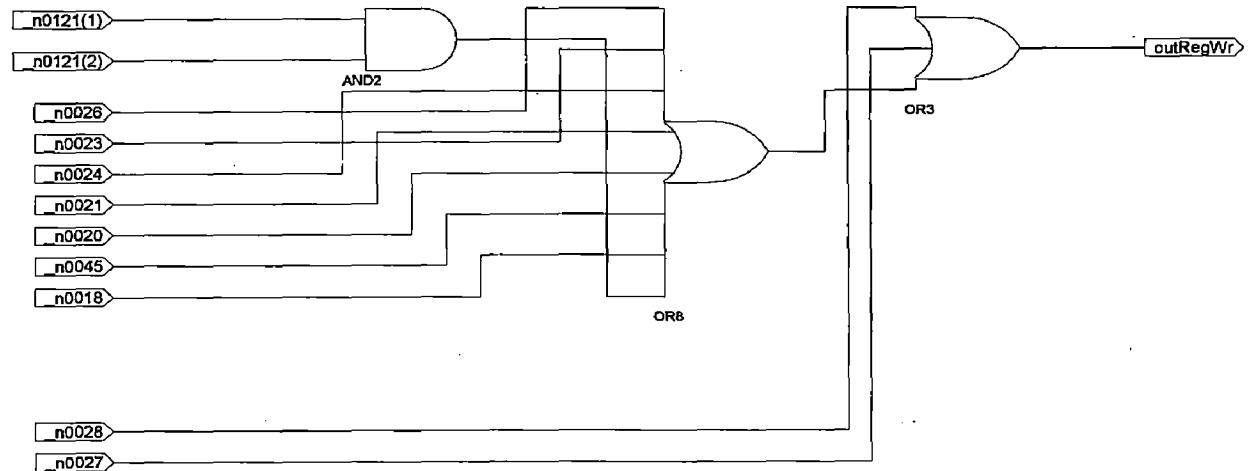


Figure B.6 RTL diagram for outregwr

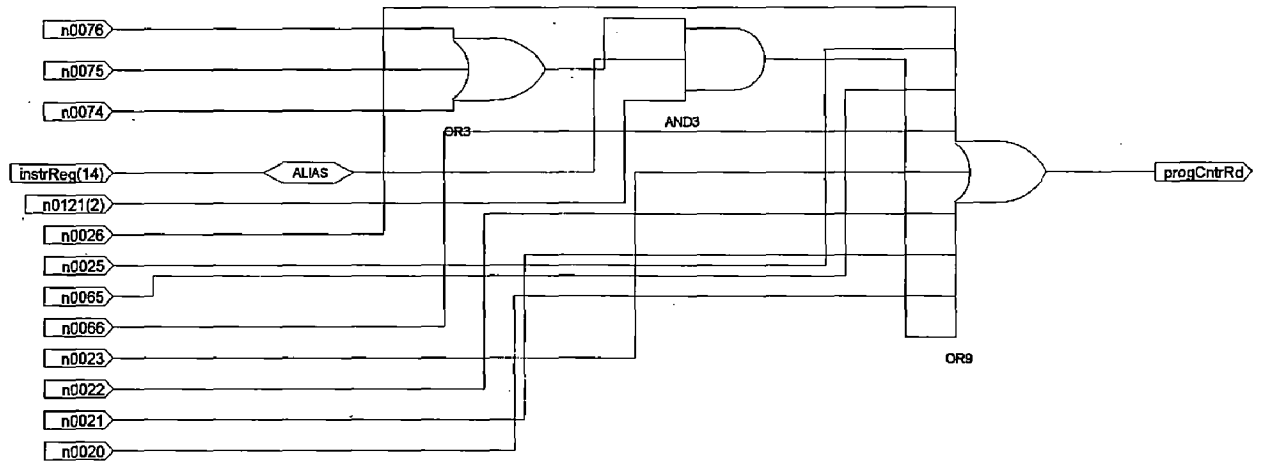


Figure B.7 RTL diagram for progcntrrd

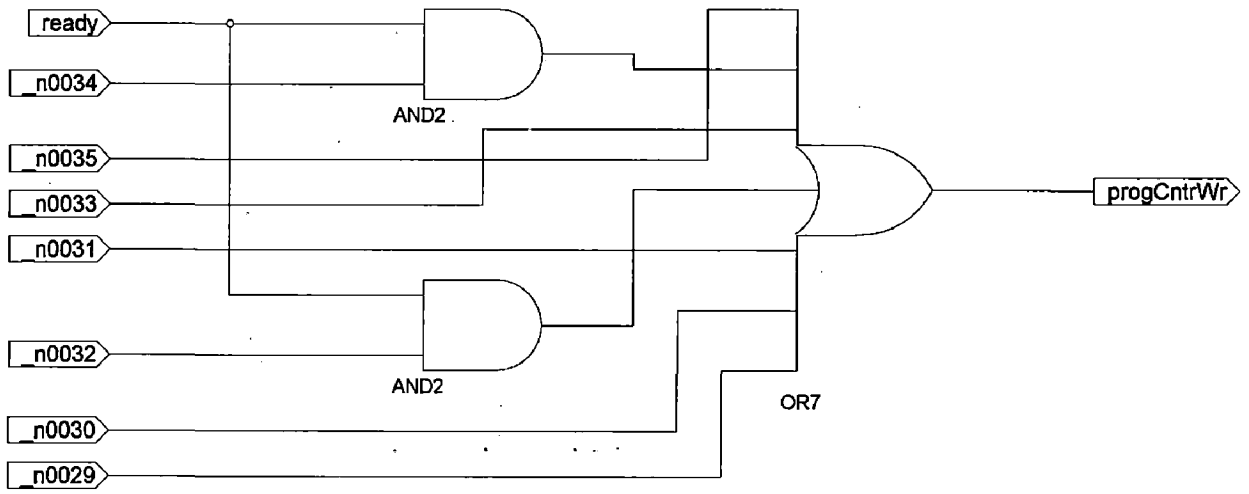


Figure B.8 RTL diagram for progcntwr

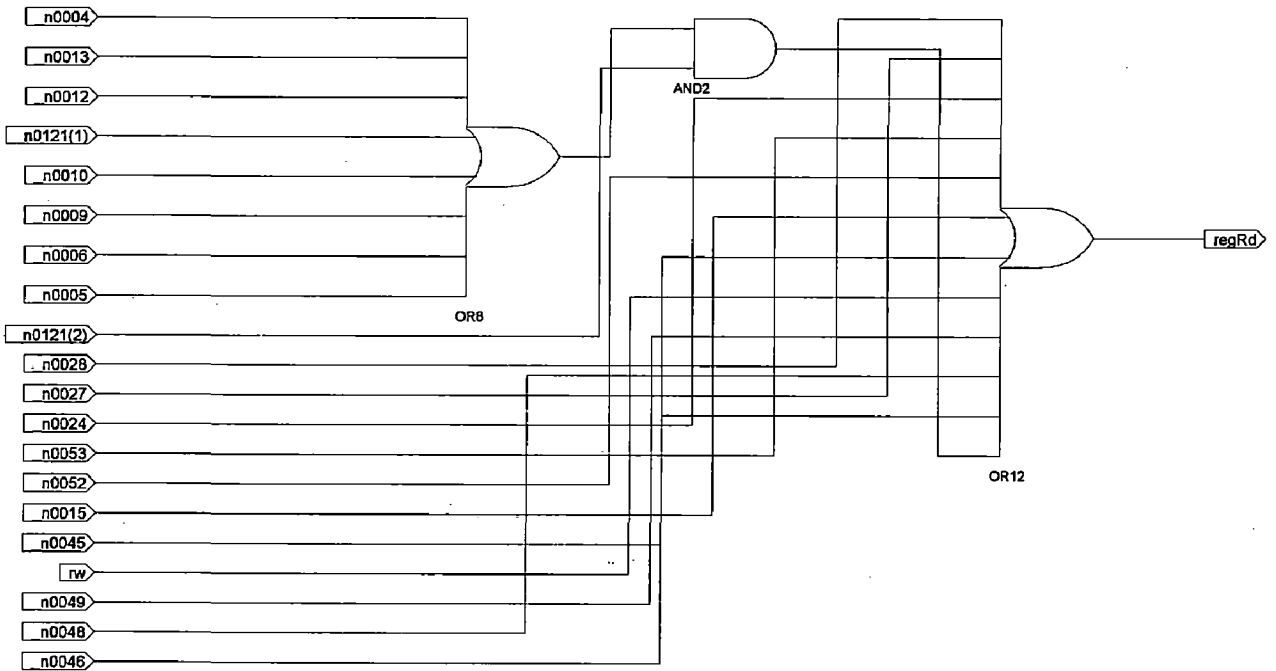


Figure B.9(a) RTL diagram for regrd

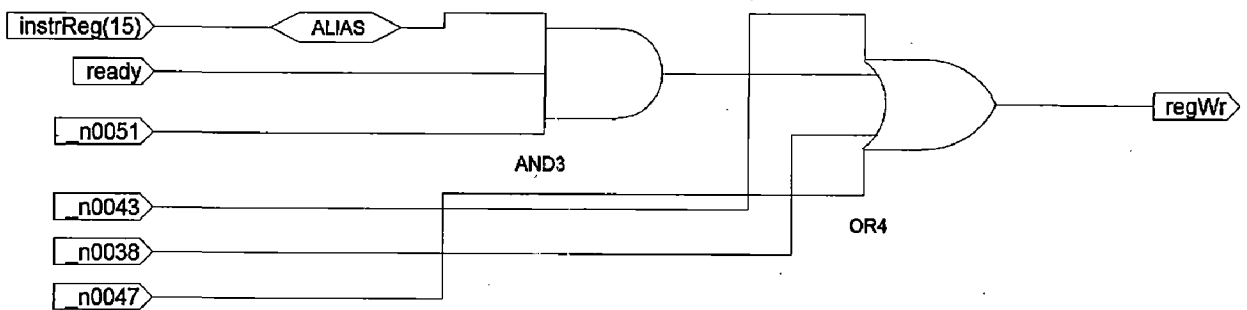


Figure B.9(b) RTL diagram for regwr

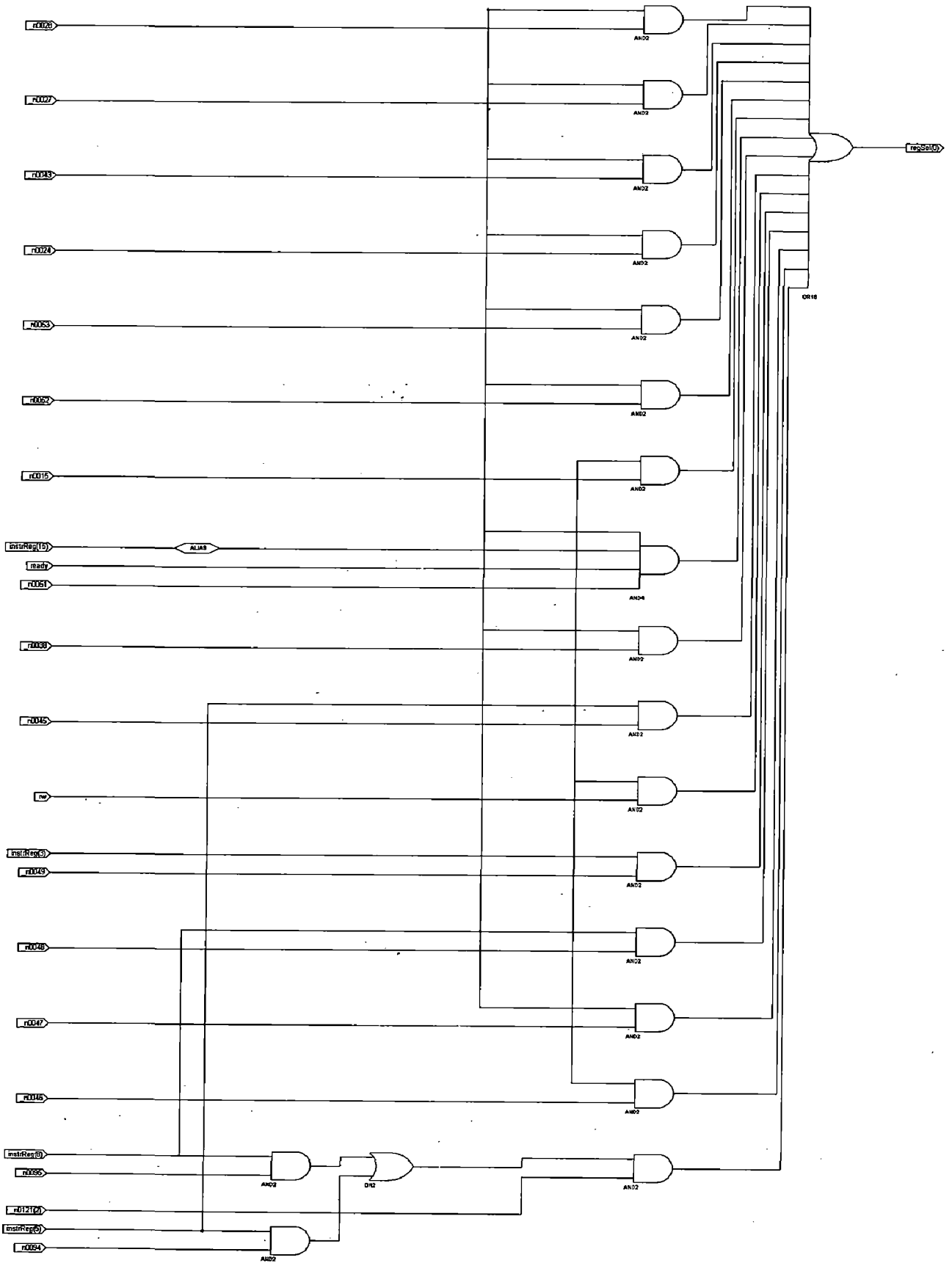


Figure B.10(a) RTL diagram for `regsel(0)`

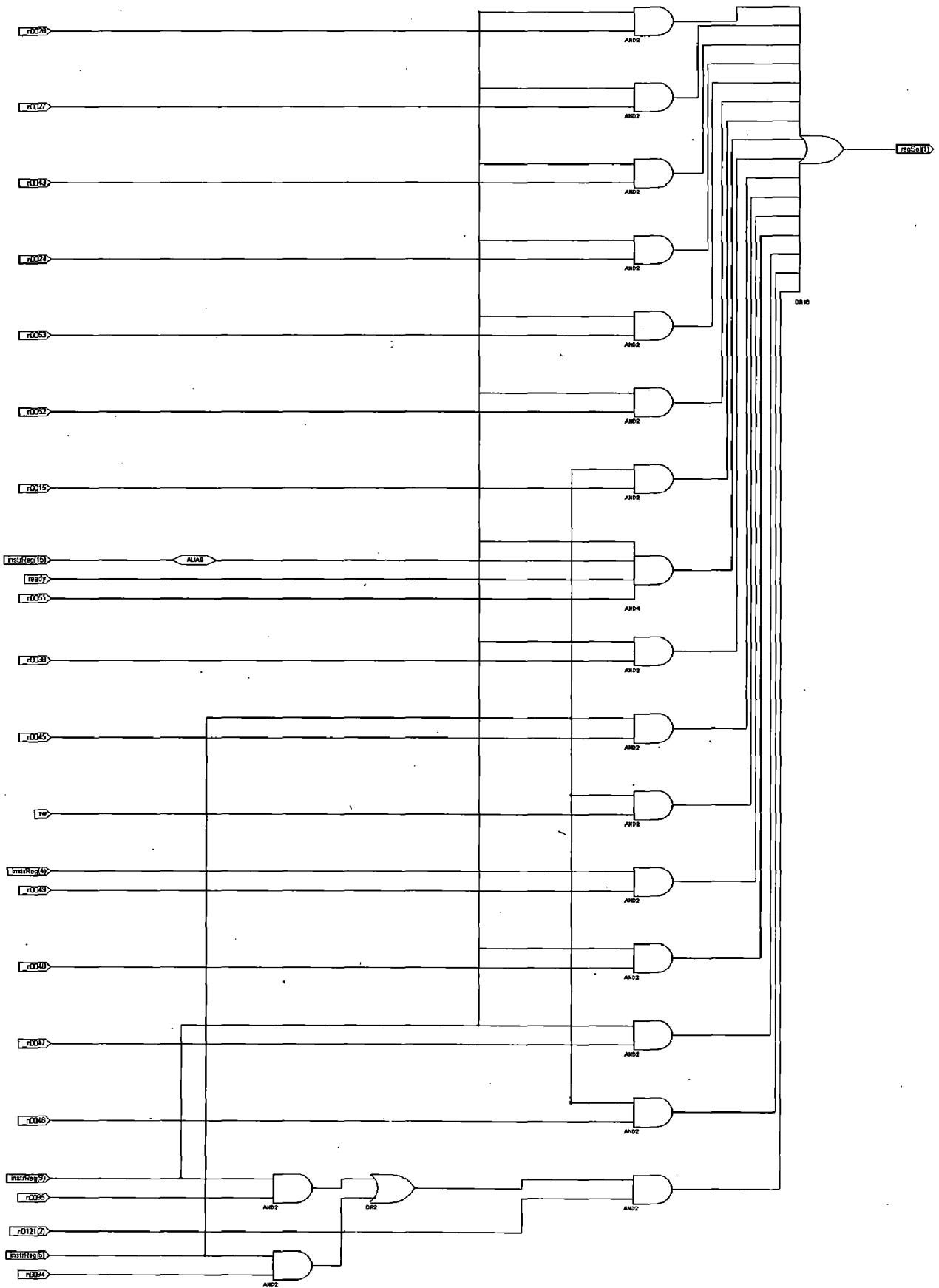


Figure B.10(b) RTL diagram for `regsel(1)`

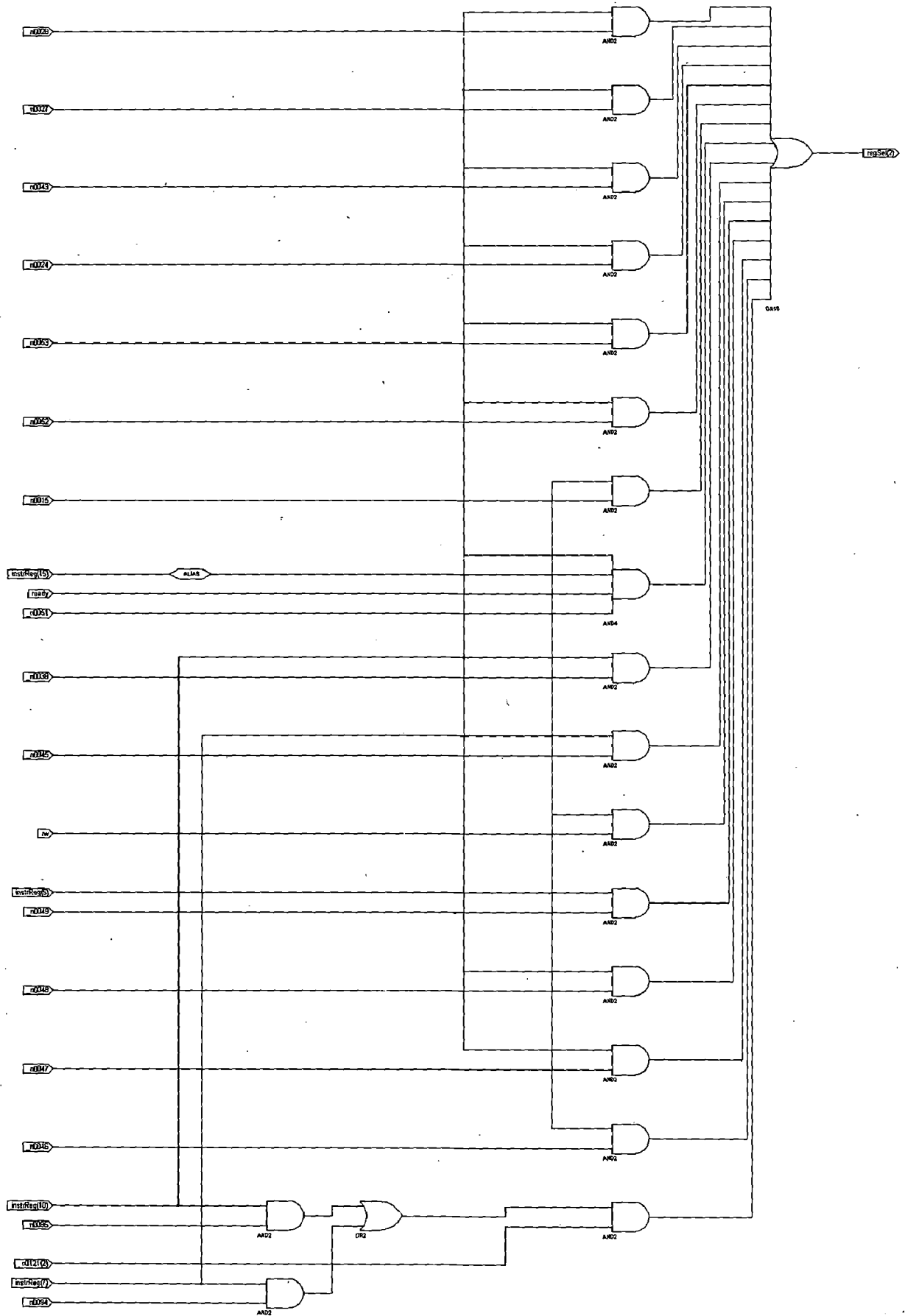


Figure B.10(c) RTL diagram for regsel(2)

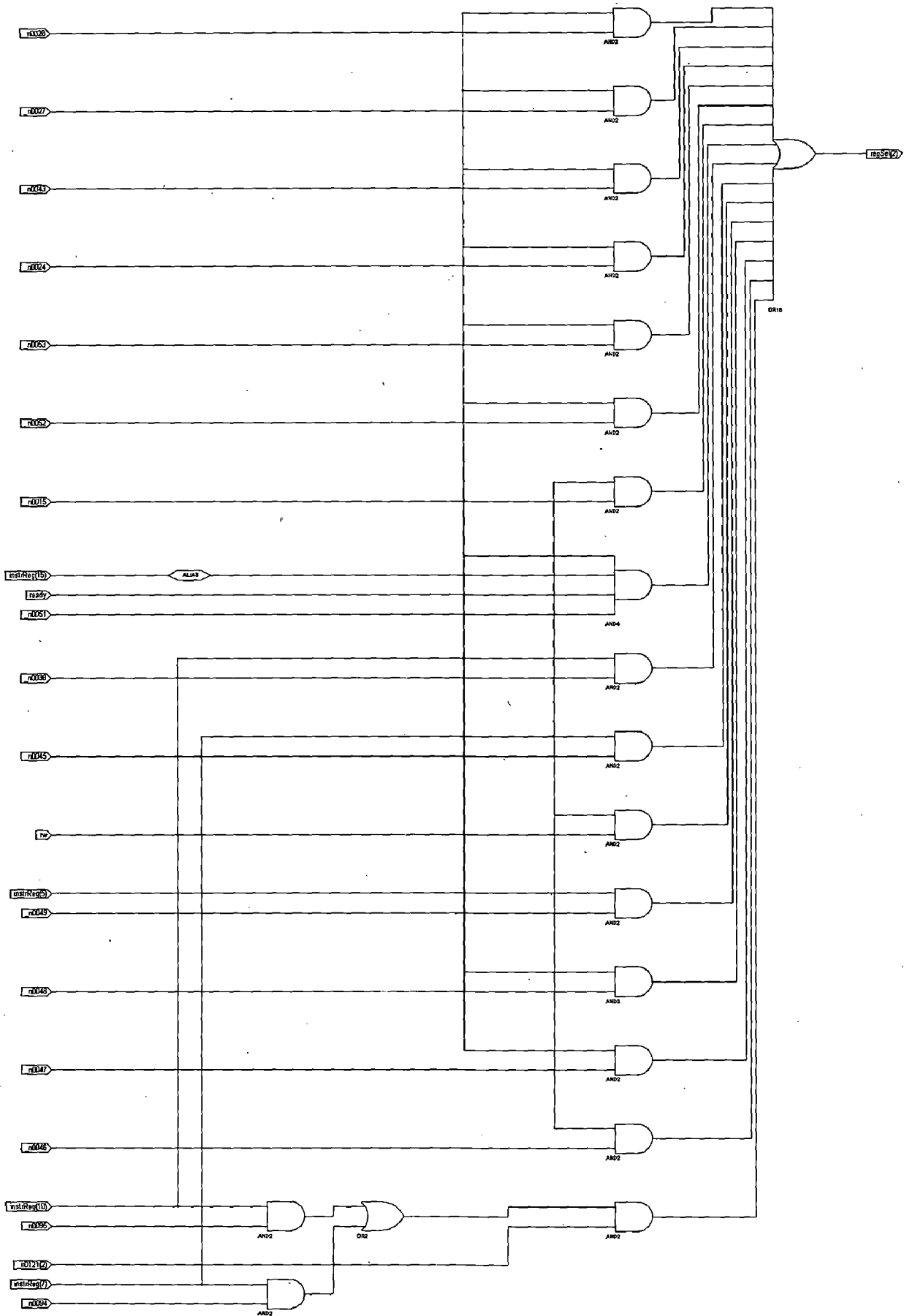


Figure B.10(c) RTL diagram for `regsel(2)`