

DESIGN OF LC-3 MICROCONTROLLER

Submitted in partial fulfilment of the Degree of
Bachelor of Technology



May-2014

By

MOHIT SINGLA (101131)

under the Supervision of

Dr. Pradeep Kumar

DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,

WAKNAGHAT

Certificate

This is to certify that project report entitled “Design of LC3 Microcontroller”, submitted by MOHIT SINGLA in partial fulfillment for the award of degree of Bachelor of Technology in Electronics and Communication Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

Date :26th May, 2014



Dr. Pradeep Kumar
Associate Professor, ECE Deptt.
JUIT Waknaghat, Solan-173234


Acknowledgement

Knowledge, energy, and time are the resources in the completion of this project but the most requisite is the proper guidance of my respected mentor, **Dr. Pradeep Kumar** (Associate Professor, Department of Electronics and Communication) to whom I extend the sincere word of thanks, for his invaluable cooperation and help throughout the project. He acted as constant source of motivation throughout the development stage of the project.

I would like to thank our H.O.D. **Dr. Sunil Bhooshan** and all the panel members, for their valuable suggestion and guidance in all the seminars and viva-voce during evaluation.

I also express my obligation to all people who helped me directly or indirectly in the completion of this project. No thanks can counter my indebtedness to my parents and family who have been with me through every thick and thin, I thank them from core of my heart.

Date : 26th May,2014



MOHIT SINGLA

TALBLE OF CONTENTS

Chapter No.	Topics	Page No.
	Certificate	I
	Acknowledgement	II
	Summary	V
	List of Figures	VI
Chapter -1	An Overview of VLSI	
	1.1 Introduction	1
	1.2 What is VLSI ?	2
	1.3 Complexity	2
	1.4 Design	3
	1.5 VLSI Chip Types	6
	1.6 Concept	7
Chapter -2	LC3 Microcontroller Design Specifications	
	2.1 Top-level module	8
	2.2 Controller	11
	2.3 Fetch	12
	2.4 Execute	14
	2.5 MemAccess	15
	2.6 Writeback	16
	2.7 Decode	17
Chapter -3	Implementation and Results	
	3.1 Fetch Block	20
	3.2 Decode Block	22
	3.3 Execute Block	23
	3.4 Writeback Block	24
	3.5 Controller	26
	3.6 Register File	27

	3.7	MemAccess Block	29
	3.8	Top Level	30
CONCLUSION			32
FUTURE WORK			32
REFERENCES			33

SUMMARY

LC3 Microcontroller is a standard microcontroller which is used in many foreign universities for various purposes. Little Computer 3 or LC-3, is a type of computer educational programming language, an assembly language, which is a type of low-level programming language. It features a relatively simple instruction set, but can be used to write moderately complex assembly programs. We dealt with only a small subset of the possible LC-3 instructions i.e. ALU Operations (AND, NOT, ADD) and some Memory Operations. Moreover, in this project we are going to be working exclusively with an un-pipelined version of the LC3. The instructions of interest for this project were chosen such that each one ends in exactly 5 clock cycles. The operation of a microcontroller is controlled by the contents of the instruction memory. The content read out, called an instruction, is a 16 bit value which causes the microcontroller to perform a specific function. To help perform the function, there would be a set of memory locations used to store values that can be shared between multiple instructions. In case of the LC3, we have 8 such locations, R0 – R7 which can be accessed for reading (using sr1 and sr2 say) and writing (using dr). LC3 The main purpose of this project is to let you start dealing with more complex designs, and become familiar with some of the elements used within a CPU.



Signature of Student

Mr. Mohit Singla

Date: 26th May , 2014



Signature of Supervisor

Dr. Pradeep Kumar

Date: 26th May , 2014

List of Figures

Figure no.	Description	Page No.
1.1	The VLSI Design funnel	2
1.2	The Y-Chart	3
1.3	VLSI Design Flow	4
1.4	A simple design flow for a microprocessor	6
2.1	Top level block diagram of LC3 microcontroller	8
2.2	Simple LC3 Schematic	10
2.3	State transition diagram	12
2.4	Fetch Block	13
2.5	Relation among Fetch, Controller, and off-chip memory	14
2.6	Execute Block	15
2.7	W_Control Signal	17
2.8	Decode Block	18
2.9	Schematic of 8 16-bit registers	19
3.1	Fetch Block Implementation	20
3.2	Fetch Block Output	21
3.3	Decode Block Implementation	22
3.4	Decode Block Output	23
3.5	Execute Block Implementation	23
3.6	Execute Block Output	24
3.7	Wrieback Block Implementation	24

3.8	Writeback Block Output	25
3.9	Controller Block Implementation	26
3.10	Controller Block Output	27
3.11	Register File Implementation	27
3.12	Register File Output	28
3.13	Mem Access block Implementation	29
3.14	MemAccess block Output	30
3.15	Top level Implementation	30
3.16	LC3 Schematic Implementation	31

CHAPTER 1

An Overview of VLSI

1.1 Introduction

The expansion of VLSI is 'Very-Large-Scale-Integration'. Here, the term 'Integration' refers to the complexity of the Integrated circuitry (IC). An IC is a well-packaged electronic circuit on a small piece of single crystal silicon measuring few mm by few mm, comprising active devices, passive devices and their interconnections. The technology of making ICs is known as 'MICROELECTRONICS'. This is because the size of the devices will be in the range of micro, submicrometers. The examples include basic gates to microprocessors, op-amps to consumer electronic ICs. There is so much evolution taken place in the field of Microelectronics, that the IC industry has the expertise of fabricating an IC successfully with more than 100 million MOS transistors as of today. ICs are classified keeping many parameters in mind. Based on the transistors count on the IC, ICs are classified as SSI, MSI, LSI and VLSI. The minimum number of transistors on a VLSI IC is in excess of 40,000.

The concept of IC was conceived and demonstrated by JACK KILBY of TEXAS INSTRUMENTS at Dallas of USA in the year 1958. The silicon IC industry has not looked back since then. A lot of evolution has taken place in the industry and VLSI is the result of this. This technology has become the backbone of all the other industries. We will see every other field of science and technology getting benefit out of this. In fact the advancements that we see in other fields like IT, AUTOMOBILE or MEDICAL, are because of VLSI. This being such an important discipline of engineering, there is so much interest to know more about this. This is the motivation for this course namely 'VLSI CIRCUITS'.

1.2 What is VLSI ?

VLSI is 'Very Large Scale Integration'. It is the process of designing, verifying, fabricating and testing of a VLSI IC or CHIP. A VLSI chip is an IC, which has transistors in excess of 40,000. MOS and MOS technology alone is used. The active devices used are CMOSFETs. The small piece of single crystal silicon that is used to build this IC is called a 'DIE'. The size of this die could be 1.5cmsx1.5cms.

1.3 Complexity

Producing a VLSI chip is an extremely complex task. It has number of design and verification steps. Then the fabrication step follows. The complexity could be best explained by what is known as 'VLSI design funnel' as shown in the Fig.1.1.

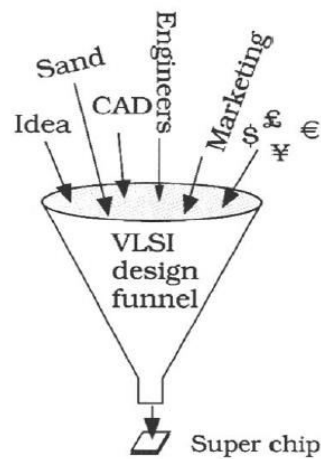


Fig. 1.1 The VLSI Design funnel

To set up facilities for VLSI one needs a lot of money. Then the design starts at a highest abstraction in designer's mind as an initial idea. Engineers using CAD tools further expand this idea. One should have good marketing information also. Then all these are dumped inside the funnel along with a pile of sand as a raw material to get the wonderful item called "the VLSI chip".

1.4 Design

A state of art of VLSI IC will have tens of millions of transistors. One human mind cannot assimilate all the information that is required to design and implement such complex chip. A design team comprising hundreds of engineers, scientists and technicians has to work on a modern VLSI project. It is important that each member of the team has clear understanding of his or her part of the contribution for the design. This is accomplished by means of the design hierarchy. Any complex digital system may be broken down into gates and memory elements by successively subdividing the system in a hierarchical manner. Highly automated and sophisticated CAD tools are commercially available to achieve this decomposition. They take very high-level descriptions of system behavior and convert them into a form that ultimately be used to specify how a chip is manufactured. A specific set of abstractions will help in describing the digital system, which is targeted for a VLSI chip. These are well depicted in the Fig.1.2 in a Y-chart. In this figure three distinct domains are marked in three directions in the form letter Y. These domains are Behavioral, Structural and Physical. The behavioral domain specifies what a particular system does. The structural domain specifies how entities are connected together to effect the prescribed behavior (or function). The physical domain finally specifies how to actually build a structure that has the required connectivity to implement the required functionality.

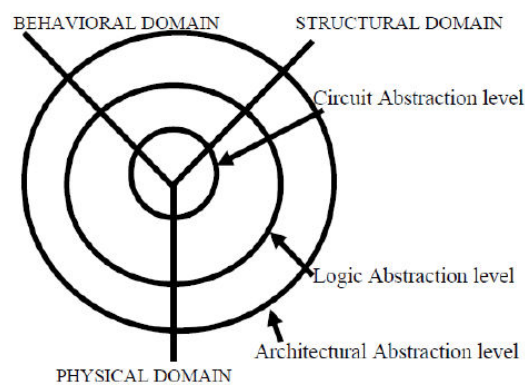


Fig. 1.2 The Y-Chart

Each design domain may be specified at a various levels of abstraction such as circuit, logic and architectural. Concentric circles around the center indicate these levels of abstraction. The design hierarchy is shown in the Fig.1.3 in the form of a flow.

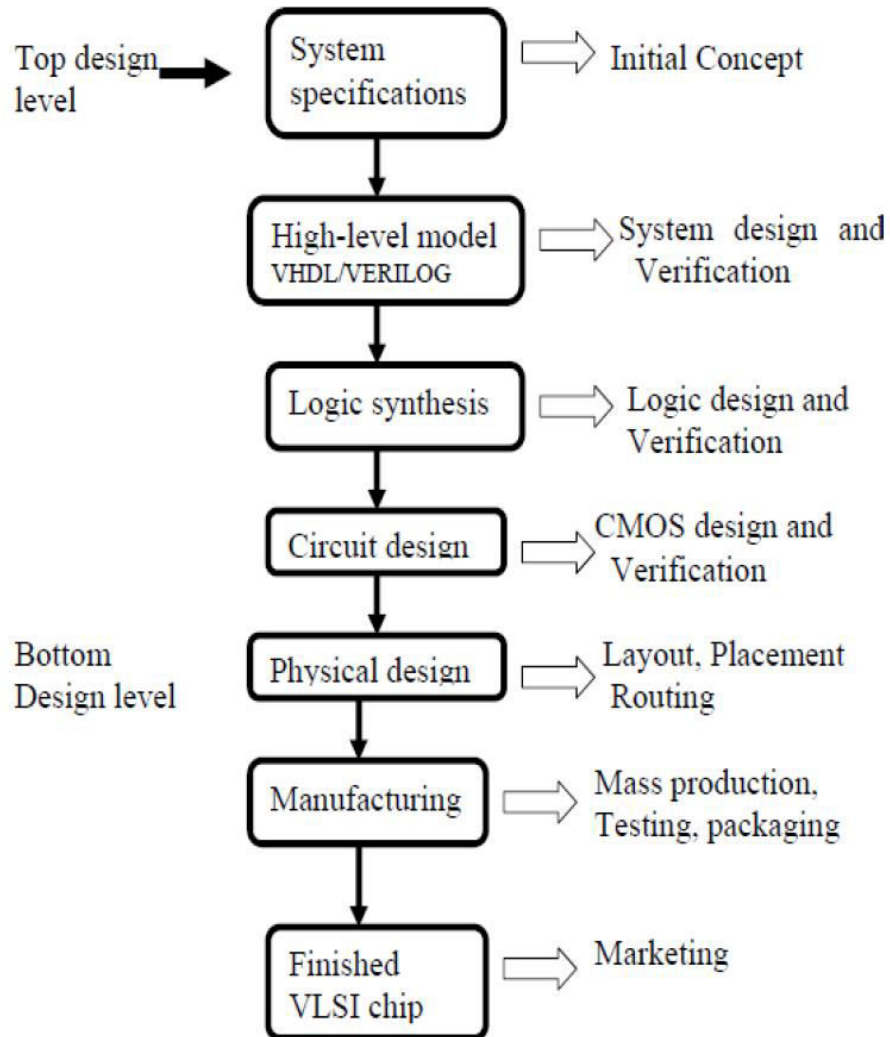


Fig. 1.3 VLSI Design Flow

- **System specifications:** is defined in both general and specific terms, such as *functions, speed, size*, etc.
- **Abstract high-level model:** contains information on the behavior of each block and the interaction among the blocks in the system.

- **Logic synthesis:** To provide the logic design of the network by specifying the primitive gates and units needed to build each unit.
- **Circuit design:** where transistors are used as switches and Boolean variables are treated as vary voltage signals.
- **Physical design:** the network is built on a tiny area on a slice of silicon.
- **Manufacturing:** a completed design process is moved on to the manufacturing line

1.4.1 Hierarchical design

- Top-down design
 - The initial work is quite abstract and theoretical and there is no direct connection to silicon until many steps have been completed
 - Acceptable in modern digital system design
 - Co-design with combining HW/SW is critical
 - Similar to Cell-based Design Flow
- Bottom-up design
 - Starts at the silicon or circuit level and builds primitive units such as logic gates, adders, and registers as the first steps.
 - Acceptable for small projects.
 - Similar to Full-custom Design Flow.
- An example of a design hierarchy in Figure 1.4
 - An instruction design of a microprocessor

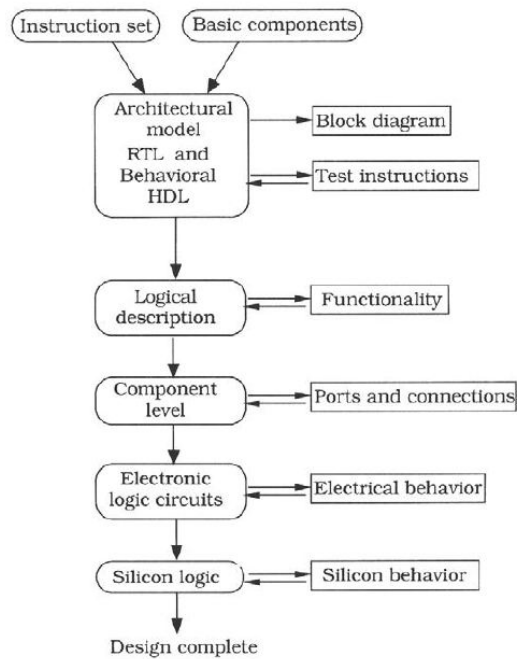


Fig. 1.4 A simple design flow for a microprocessor

1.5 VLSI Chip Types

At the engineering level, digital VLSI chips are classified by the approach used to implement and build the circuit .

- **Full-custom Design:** where every circuit is custom designed for the project
 - Extremely tedious
 - Time-consuming process
- **Application-Specific Integrated Circuits (ASICs):** using an extensive suite of CAD tools that portray the system design in terms of standard digital logic constructs.
 - Including state diagrams, functions tables, and logic diagram
 - Designer does not need any knowledge of the underlying electronics or the physic of the silicon chip
 - Major drawback is that all characteristics are set by the architectural design
- **Semi-custom Design:** between that of a full-custom and ASICs
 - Using a group of primitive predefined cells as building blocks, called cell library.

1.6 Concept

VLSI should be thought of as a single discipline that deals with the conception, design and manufacture of complex ICs. Carver Mead is the gentle man who did pioneering work towards VLSI in 1970s. He came out with the standard definition with regard to the formation of a MOS transistor on silicon, which states that ‘when polysilicon cuts across the diffusion, a transistor is formed at the intersection’. Thus he observed that the digital IC could be viewed as a set of geometrical patterns (polygons) on every layer that is going to be integrated on the silicon surface. Thus an IC will comprise of innumerable polygons of conducting (metal and polysilicon), semi conducting (silicon) and non-conducting (insulator such as SiO₂) layers at various levels of integration. Groups of patterns represent different logic functions and these are replicated throughout the IC. Thus the complexity is broken down using the concept of repeated patterns that were fitted together in a structured manner.

The size of the transistor has been reducing ever since the concept of IC was conceived since 1958. In 1970 Gordon Moore predicted the growth of microelectronics in terms of number of transistors that could be fabricated on a chip. He projected that the number of transistors would get doubled every 18 to 24 months. This has been established as ‘MOORE’S LAW’. The silicon industry is facing a tough challenge to keep the pace with the law. On the other hand it is not possible to manufacture a functional design because of defects in the silicon crystal structure that cannot be avoided. The larger the area of the circuit, the higher the probability that a defect will occur. Even a single bad transistor or connection (because of the defect) would make the chip unusable. Therefore the philosophy is to keep the overall size of the chip small.

CHAPTER 2

LC3 Microcontroller Design Specifications

The microcontroller is a simplified version of the original LC3 microcontroller. Specifically, four simplifications are considered as follows:

- A smaller instruction set: the ISA you need to implement does NOT contain the control instructions RTI and TRAP. All other instructions must be implemented.
- No off-chip memory: The instructions of the program are assumed to be in the cache.
- The programs consist of valid instructions ONLY, i.e., you do not have to perform error checking to detect bad instructions
- No overflow detection is required.

2.1 Top-level module:

To begin, the top level block diagram of LC3 Microcontroller is shown in fig. 2.1

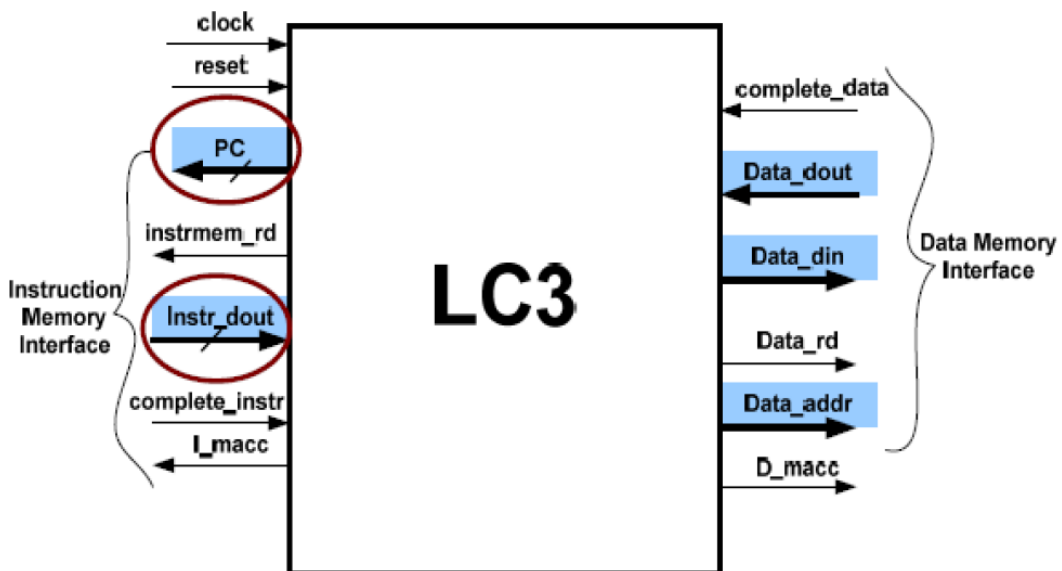


Fig. 2.1 Top level block diagram of LC3 microcontroller

The inputs and outputs to this design are:

Inputs:

- clock (**1 bit**)
- reset (**1 bit**)
- complete_instr or complete (**1 bit**) : Signal to indicate completion of read/write
- Instr_dout or dout (**16 bits**) : Corresponds to the instruction from the Instruction Memory i.e. Data-Out lines from Memory.

Outputs:

- PC or addr (**16 bits**) : This corresponds to an address to the Instruction Memory
- instrmem_rd or rd (**1 bit**) : This signal enables a read from the Instruction Memory for a fetch. i.e. Memory signal to indicate read or write
- Data_addr (**16 bits**) : Corresponds to the address sent to the Data Memory for reads from it. Ignore
- Data_din (**16 bits**) : Corresponds to the values that need to be written to Data Memory which would correspond to stores. Ignore
- Data_rd (**1 bit**) : This signal enables a read from the Data Memory. If this signal is 0 then a write to Data Memory is enabled. Ignore
- I_macc and D_macc : These will be used in the future projects to distinguish between Instruction and Data memory access phases. Ignore

As stated earlier, we are going to deal with only a smaller subset of the possible LC3 instructions *i.e.* ALU Operations (AND, NOT, ADD) and the one of the Memory operations called LEA (Load Effective Address). The reason for the use of this subset is to provide an introduction to the signals of importance in the datapath of the LC3. Moreover, in this project we are going to be working exclusively with an un-pipelined version of the LC3. Additionally, the instructions of interest for this project have been chosen such that each one ends in exactly 5 clock cycles.

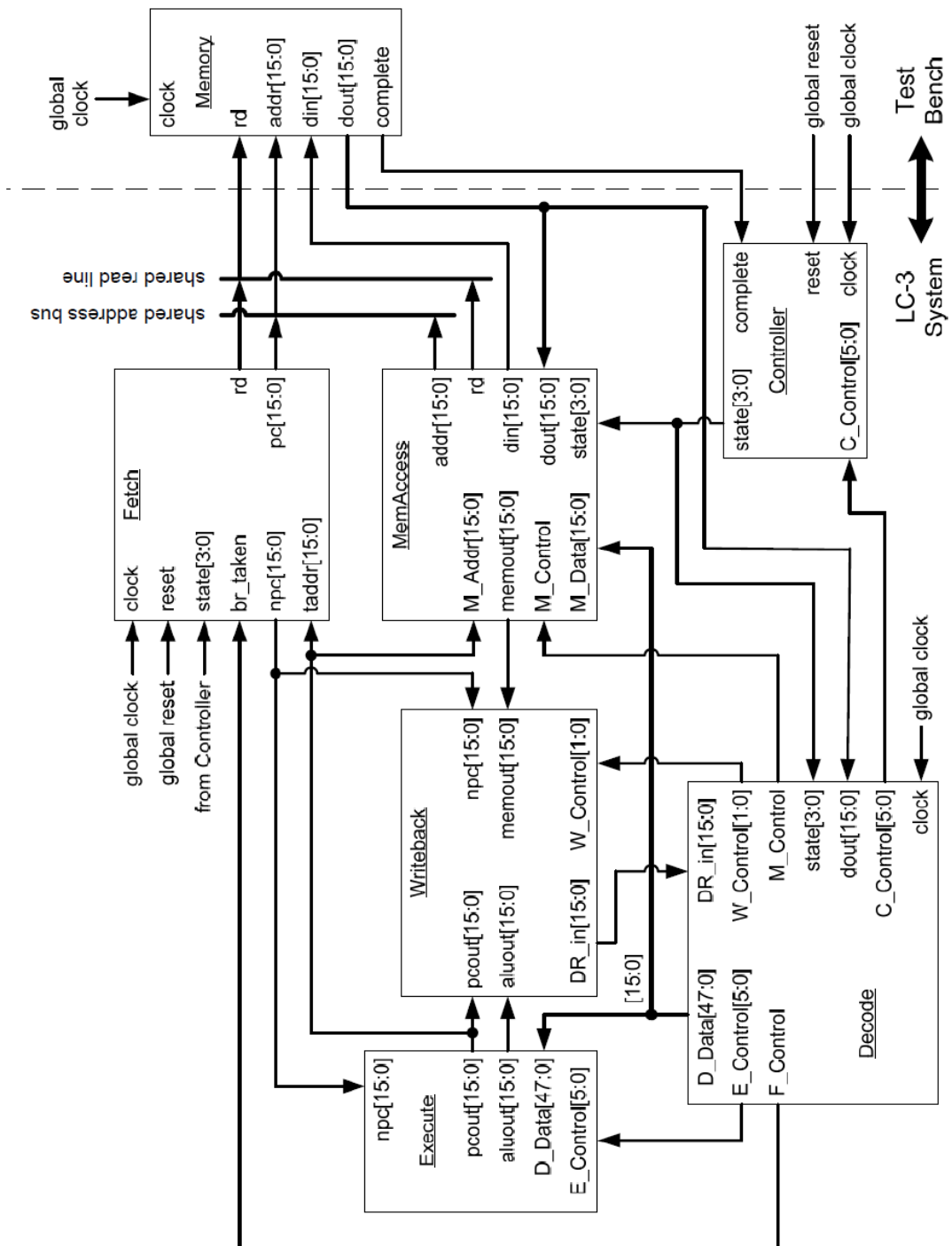


Fig. 2.2 Simple LC3 Schematic

The top-level module is instantiated in the test-bench along with the memory.

Special Signals:

The **SimpleLC3** module should be connected exactly as shown in the schematic. Note that there is a shared read-line and a shared address-bus for the memory, which means that these signals will be driven from two sources. In addition, note that **VSR2** field (which in this schematic is the least significant 16 bits of the the **D_Data** signal to the **Execute** block) goes into the **MemAccess** block as the **M_Data** signal. All other signals are simple inputs and outputs.

2.2 Controller

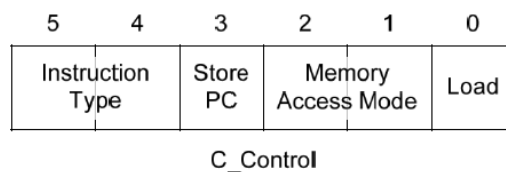
Inputs :

- clock : System clock
- reset : System reset
- C_Control (7 bits) : Control from decode
- complete : Complete from Memory

Outputs:

- state (4 bits) : System state

The Controller module is a finite state machine that controls the dataflow and therefore the execution of all the instructions in the microcontroller. The state transition diagram sketch is given below, in which the vertices represents states with the corresponding operations described inside. The transitions are denoted by the edges. The condition of each transition is determined by the current state and/or input signal C_Control generated by the decoder module. The C_Control can be broken down further into 4 fields as follows.



The self-pointing edges are used for Project 2 to cope with the memory latency. Such looping transitions only occur when the **complete** signal is zero, which never happens in this Project .State transitions occur and only occur at positive edges of the clock signal. When the **reset** signal is high, the next-state should be the “Fetch Instruction” state.

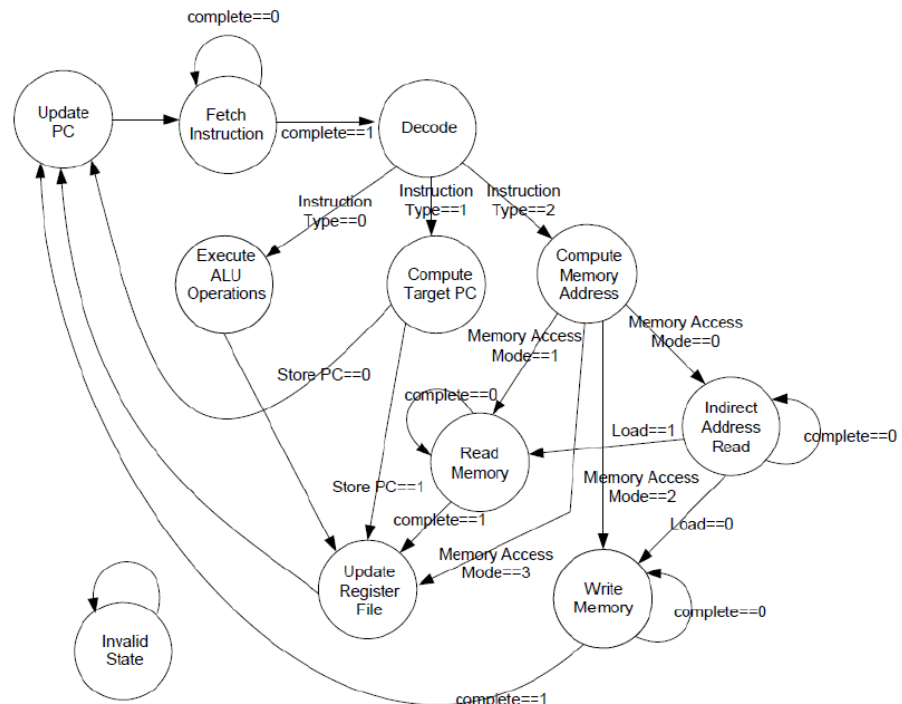


Fig. 2.3 State transition diagram

2.3 Fetch

Inputs:

- clock (1 bit) : system clock
- reset (1 bit) : system reset
- br_taken (1 bit) : signal from decoder, 1 means branch taken
- taddr (16 bits) : target address of control instructions
- state (4 bits) : system state for controller

Outputs:

- pc , npc (16 bits) : current PC and next PC i.e pc+1
- rd (1 bit) : memory read control signal

Fetch module is used to generate the program counter, which contains the address of the instruction to be fetched. The PC should be updated on the rising edge of the clock. Also, the PC should be updated only when the system is in the “Update PC” state, as determined by the Controller block. The signal **rd** should be high-impedance during the “Read Memory”, “Write Memory”, and “Indirect Address Read” states, because the MemAccess block will drive the shared memory bus during these cycles. In all other states, this signal should be high. **pc** is the memory address and should be high-impedance at the same times that **rd** is high-impedance. The first program instruction is located at the address 16’h3000. Therefore, **pc** should be set to 16’h3000 when **reset** is high. The block diagram of Fetch module is shown below

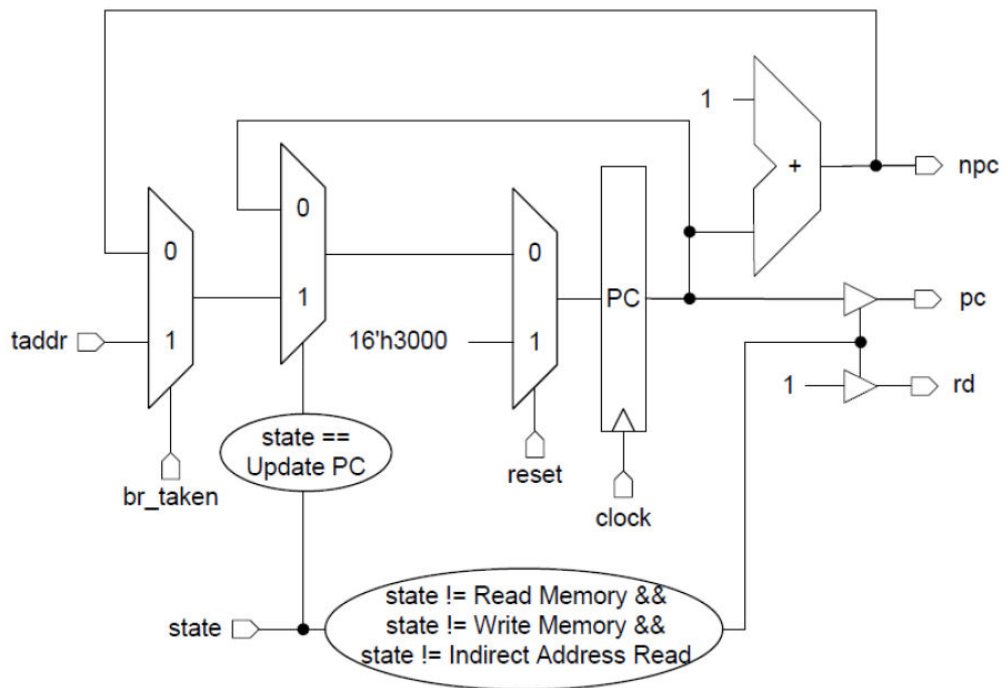


Fig. 2.4 Fetch block

The relation among Fetch, Controller, and off-chip memory module is shown below.

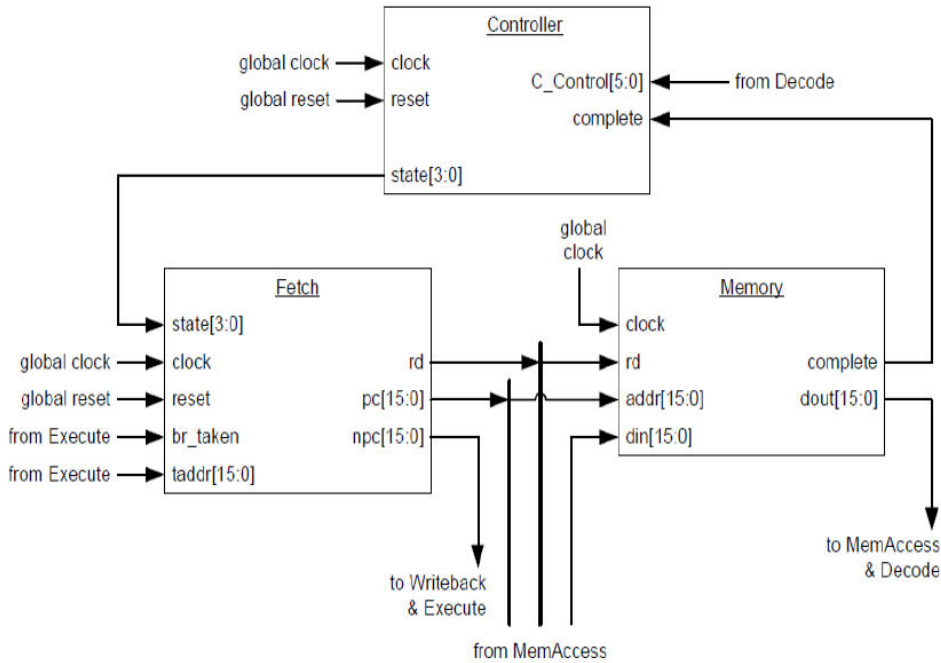


Fig. 2.5 Relation among Fetch, Controller, and off-chip memory

2.4 Execute

Inputs:

- E_control (6 bits) : control signals from decode
- D_data (48 bits) : data from decode
- npc (16 bits) : next pc from fetch

Outputs:

- aluout (16 bits) : output of ALU
- pcout (16 bits) : output of the address computation adder
-

Execute module performs the arithmetic and logical instructions, target PC computation, and memory address computation. The **E_Control** input is an aggregate of the **ALU Operation Sel**, **OP 2 Sel**, **PC Sel 1**, and **PC Sel 2**. The **D_Data** input is an aggregate of the **IR**, **VSR1** and **VSR2** signals. The block diagram is given below with the **ALU**. Note that overflow checking is not being done, so the **alucarry** output of the ALU is ignored.

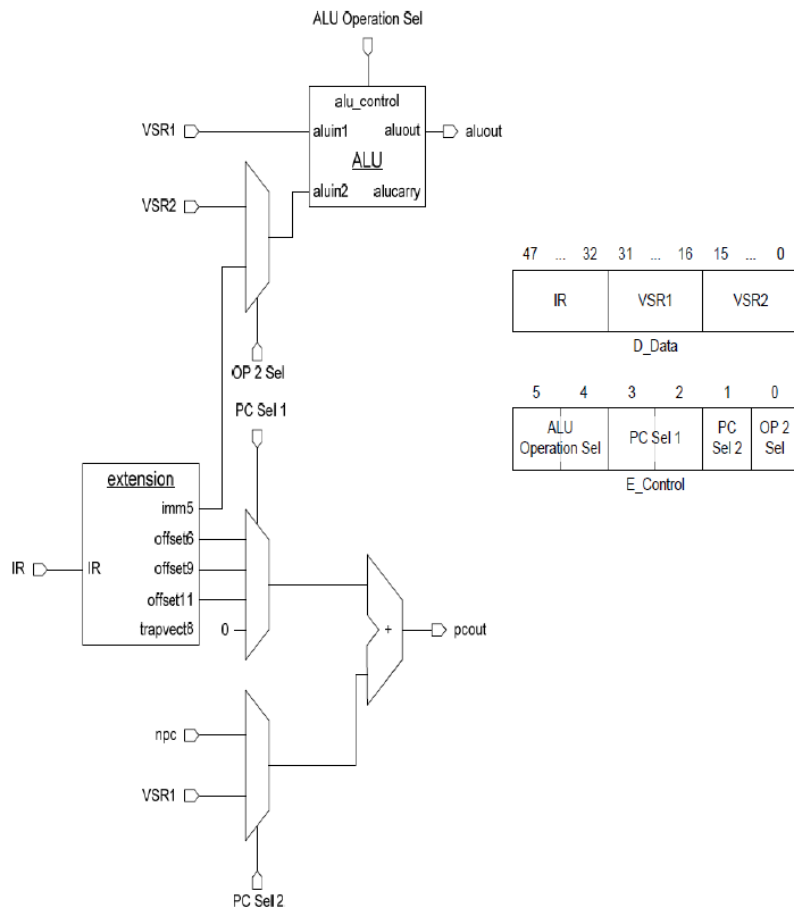


Fig. 2.6 Execute Block

2.5 MemAccess

Inputs :

- state (4 bits) : System State from Controller
- M_Control (1 bit) : control signal to indicate address from dout
- M_Data (16 bits) : Data for store operations
- M_Addr (16 bits) : Address for load/store operations
- dout (16 bits) : Data – out lines from memory

Outputs :

- **addr** (16 bits) : Address lines to memory
- **din** (16 bits) : Data-in lines to memory
- **rd** (1 bit) : Memory signal to indicate read or write

The **MemAccess** block is the master of the shared memory bus during the Read Memory, Write Memory, and Read Indirect Address states. It should setup the memory bus lines as follows:

- Read Memory – **rd** should be 1 and **din** doesn't matter. **addr** should be set to either **M_Addr** or **dout**, depending on **M_Control**. **addr** should be set to **dout** in this state only if the opcode shows an LDI operation.
- Write Memory – **rd** should be 0 and **din** should be **M_Data**. **addr** should be set to either **M_Addr** or **dout**, depending on **M_Control**. **addr** should be set to **dout** in this state only if the opcode shows an STI operation.
- Read Indirect Address - **rd** should be 1 and **din** doesn't matter. **addr** should be set to **M_Addr**.

The **memout** signal should always pass the value of **dout** through to the **Writeback** block.

2.6 Writeback

Inputs :

- **aluout**, **memout**, **pcout**, **npc** (16 bits) : Possible data to store
- **W_Control** (2 bits) : control signal to choose what will be written

Outputs:

- **Dr_in** (16 bits) : data that will be stored in the register-file

The **Writeback** block should set the **DR_in** lines to the value to be written into the register-file. This value is selected from the following four choices:

- **aluout** – The output of the ALU in the **Execute** block
- **pcout** – The computed memory address output of the **Execute** block

- **npc** – The next value of the program counter from the **Fetch** block
- **memout** – The value read from memory, from the **MemAccess** block

The **W_Control** signal will be used to select between these possibilities. The schematic is shown below.

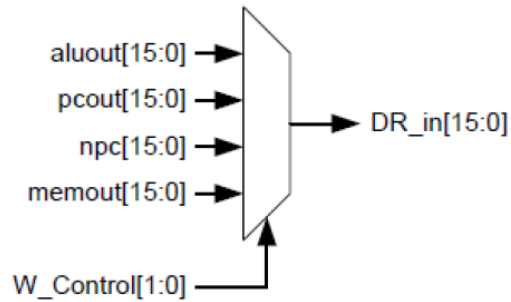


Fig. 2.7 W_Control Signal

2.7 Decode

Inputs :

- clock (1 bit) : Global system clock
- state (4 bits) : System State from Controller
- dout (16 bits) : Data – out lines from memory
- DR_in (16 bits) : Data to be written to the Register file

Outputs :

- M_Control (1 bit) : MemAccess control line
- W_Control (2 bits) : Writeback control lines
- C_Control (6 bits) : Controller control lines
- E_Control (6 bits) : Execute control lines
- D_Data (48 bits) : Data for Execute and MemAccess blocks
- F_Control (1 bit) : Fetch control line

The Decode block contains the logic illustrated in the schematic below. It contains an instruction register (IR) that stores the current instruction during the Decode state. It contains

a program status register (PSR) that stores the status of the last value written to the register file (positive, negative, or zero) and is update only on the Update Register File state. Lastly, it contains a register file that can read two locations on one cycle and write to one location in the same cycle. However, the register file writes only during the Update Register File state.

Based on the contents of IR and PSR, the decode block generates all of the control signals for the other blocks (C_Control, M_Control, W_Control, E_Control, and F_Control) as well as the source and destination addresses in the register file (sr1, sr2, and dr). Note, however, that the “instruction type” field of the C_Control signal must be valid during the Decode state and will therefore not be valid if this field is computed from the contents of IR. Therefore, the “instruction type” field is computed from the memory output, which makes it valid during the Decode state (but not necessarily the states after Decode).

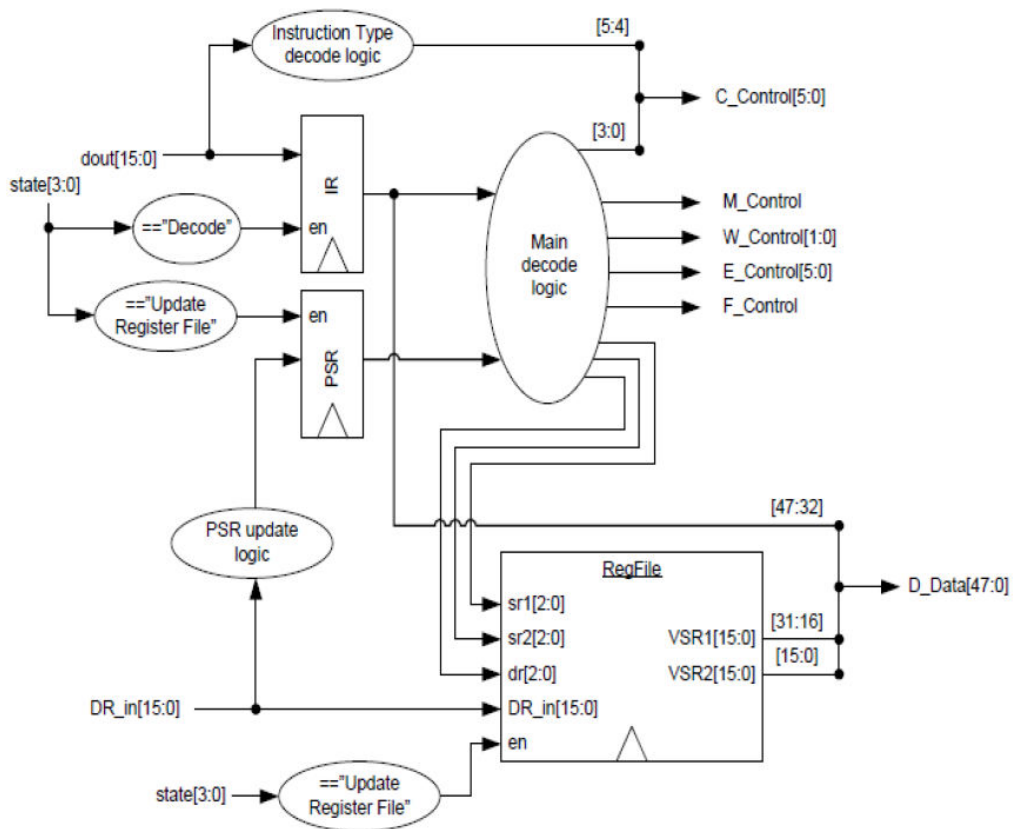


Fig. 2.8 Decode Block

The **RegFile** block inside the **Decode** block can be conceptually thought of as the schematic below. In the schematic, the outputs of eight 16-bit registers fan-out to two 8-to-1 MUXes,

which are used to determine the **VSR1** and **VSR2** signals, depending on the **sr1** and **sr2** select lines. The **DR_in** input fans-out to the data-inputs of all eight registers. Each register has an enable input that determines if it will load the input value, and these enable inputs are connected to a 1- to-8 decoder that passes the master enable signal to one register, depending on the **dr** input.

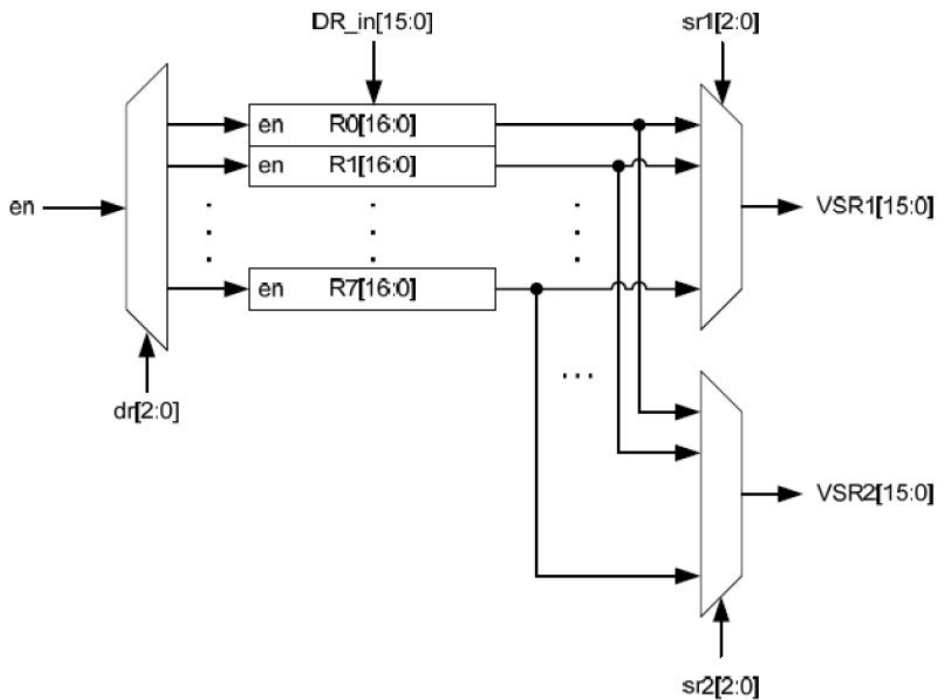


Fig. 2.9 Schematic of 8 16-bit registers

You may want to implement the **RegFile** with a memory, rather than eight individual registers. The code will be much simpler and easy to understand. However, the waveform capture formats typically don't store the values in memories. If you want to view the contents of your register file in a waveform viewer, you may want to have a set of eight assign statements, such as the following:

```
assign R0=mem[0];
assign R1=mem[1];
...
```

This approach will not affect your synthesis results. You should still get $16 \times 8 = 128$ flip-flops when synthesizing.

CHAPTER 3

Implementation and Results

3.1 Fetch Block

3.1.1 Implementation

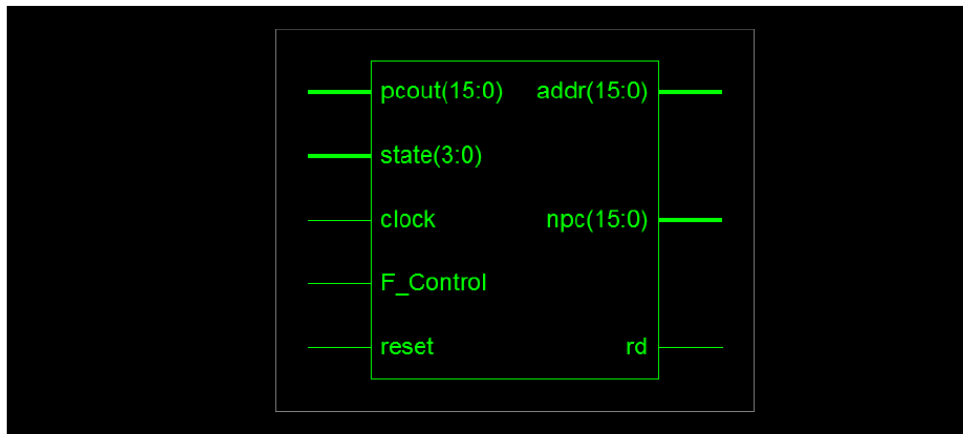


Fig. 3.1 Fetch Block Implementation

3.1.2 Results

Input data :

```
initial
begin
    pcout = 16'h4000;
    F_Control=0;
end
```

```
initial
begin
    reset=1;
    #10 reset = 0;
    #100 reset = 1;
```

```

#20 reset = 0;

end

initial
  clock=0;

always
  #5 clock = ~ clock;

initial
  begin
    state = 4'b0001;
    #10 state = 4'b1010;
    #30 state = 4'b1010;
    #55 state = 4'b0111;

  end
end

```

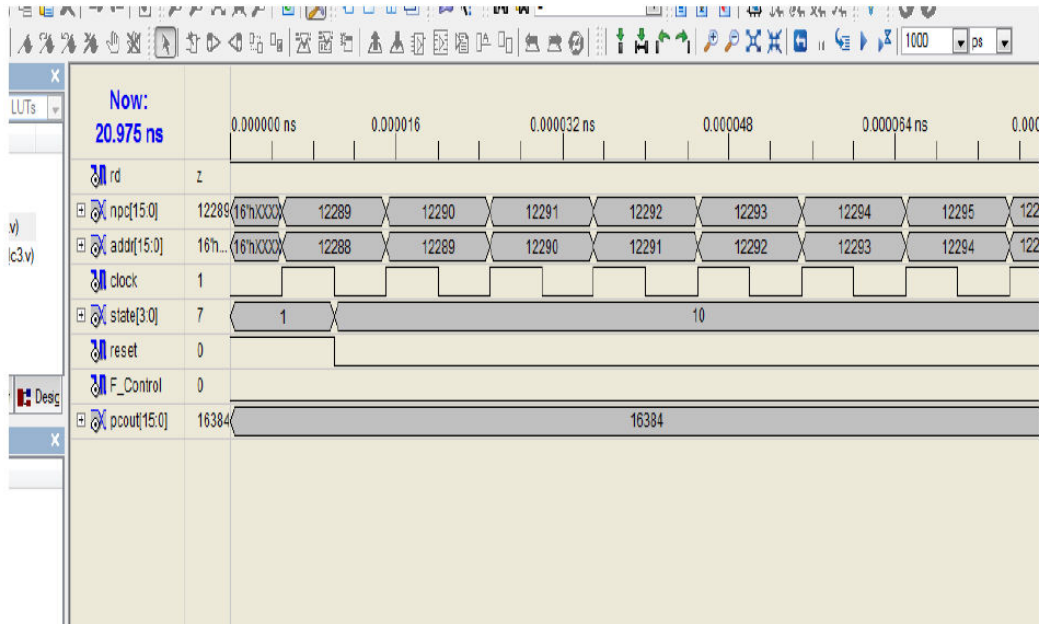


Fig.3.2 Fetch Block Output

3.2 Decode Block

3.2.1 Implementation

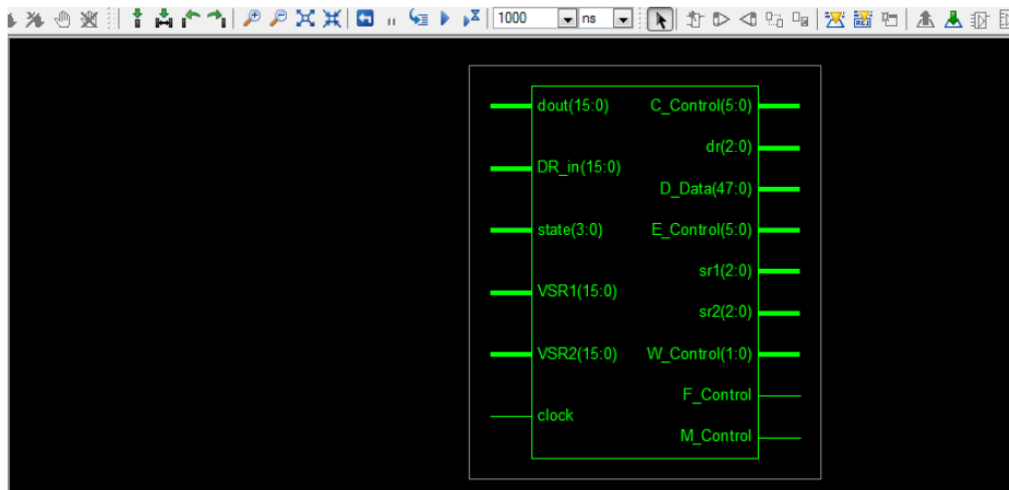


Fig. 3.3 Decode Block Implementation

3.2.2 Results

```
Input data :
  initial
    clock = 0;

  always
    #5 clock = ~clock;

  initial
    begin
      state = 4'b0010;
      VSR1 = 16'h2345;
      VSR2 = 16'h5698;
      #25 dout = 16'h9637;
      DR_in = 16'h9783;
      #15 state = 4'b1001;
      #10 state = 4'b0010;
      dout = 16'h5785;
    end
```

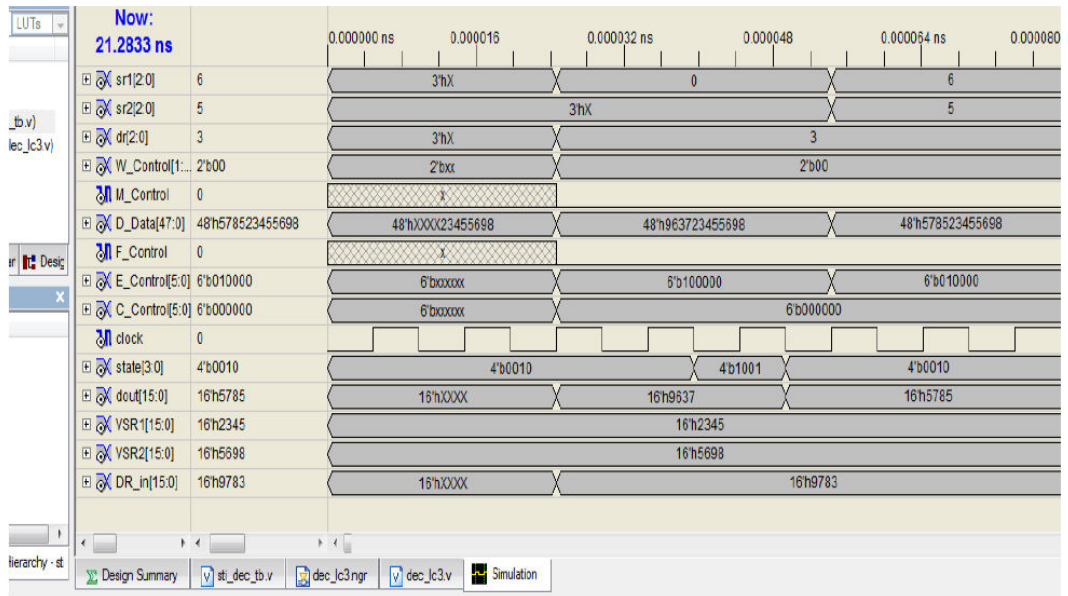


Fig.3.4 Decode Block Output

3.3 Execute Block

3.3.1 Implementation

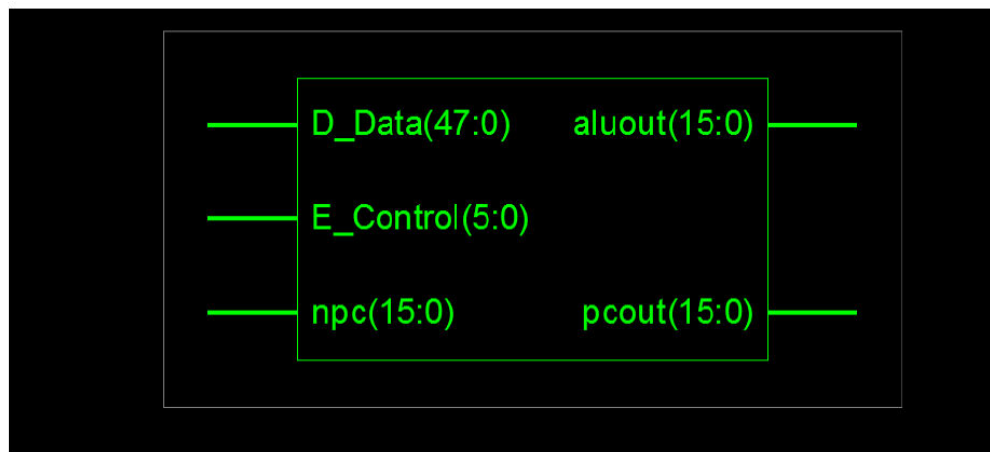


Fig. 3.5 Execute Block Implementation

3.3.2 Results

Input data :

```
begin
    npc = 16'h7849;
    D_Data = 48'h963723455698;
    E_Control = 6'b000110;
    #20 D_Data = 48'h578523455698;
    E_Control = 6'b010111;
end
```

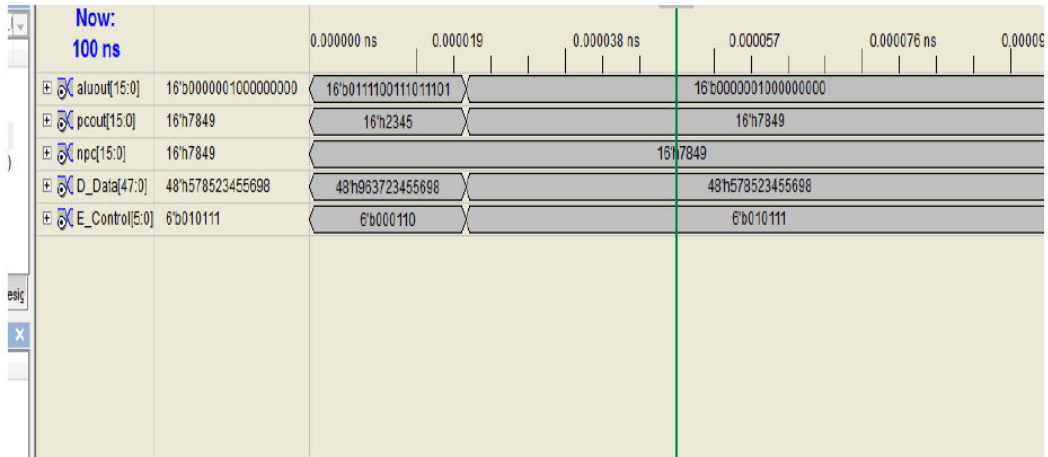


Fig.3.6 Execute Block Output

3.4 Writeback Block

3.4.1 Implementation

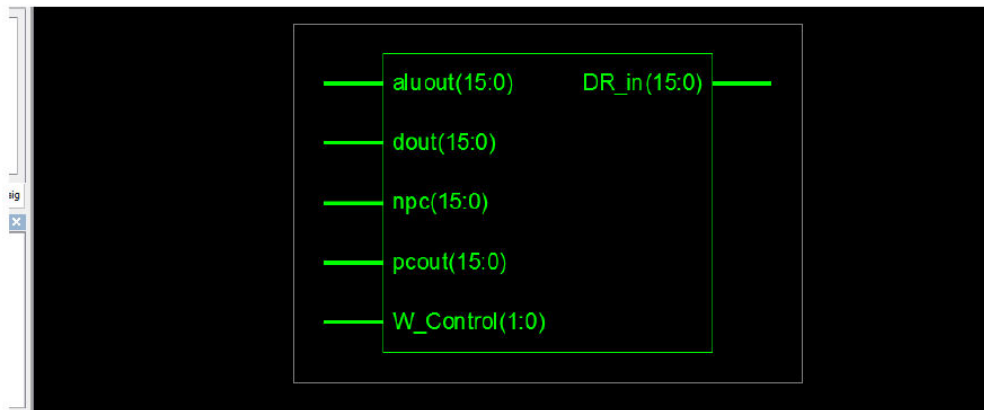


Fig. 3.7 Writeback Block Implementation

3.4.2 Results

Input data :

```
begin
    W_Control = 2'b00;
    aluout = 16'h0000;
    dout = 16'h0000;
    pcout = 16'h0000;
    npc = 16'h0000;
    #15 aluout = 16'h2467;
    dout = 16'h8463;
    pcout = 16'h6403;
    npc = 16'h6371;
    #20 W_Control = 2'b01;
    #25 W_Control = 2'b10;
end
```

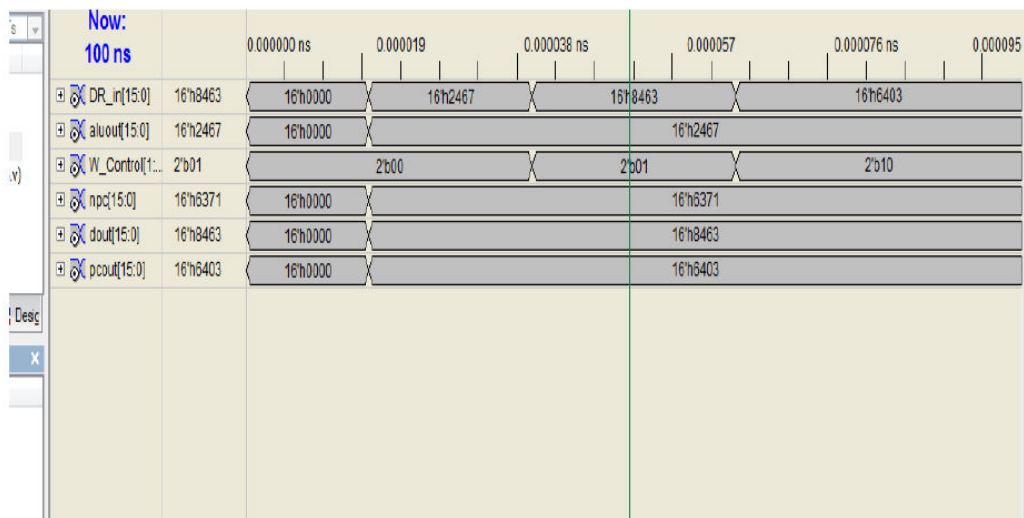


Fig.3.8 Writeback Block Output

3.5 Controller

3.5.1 Implementation

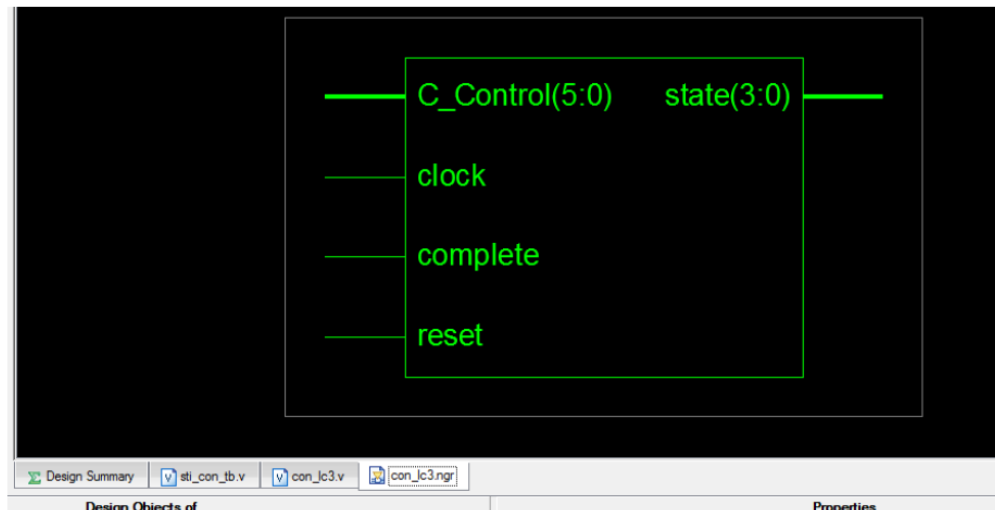


Fig. 3.9 Controller Block Implementation

3.5.2 Results

Input data :

Initial

```
clock = 0;
```

initial

```
begin
```

```
reset = 1 ;
```

```
# 10 reset = 0;
```

```
# 70 reset = 1;
```

```
end
```

always

```
#5 clock = ~clock;
```

initial

```
begin
```

```
C_Control = 6'bxxxxxx;
```

```
complete = 1;
```

```
end
```

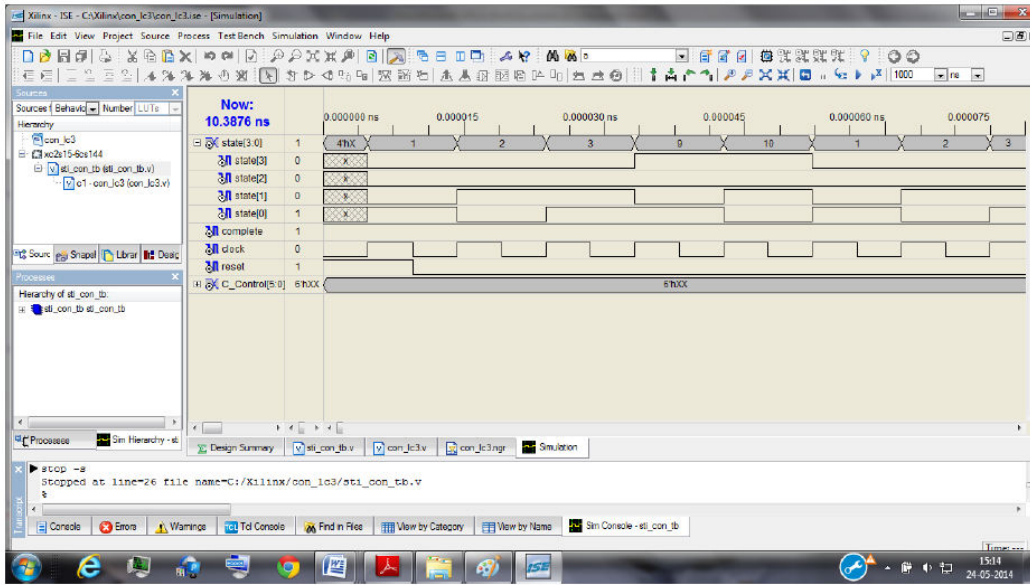


Fig.3.10 Controller Block Output

3.6 Register File

3.6.1 Implementation

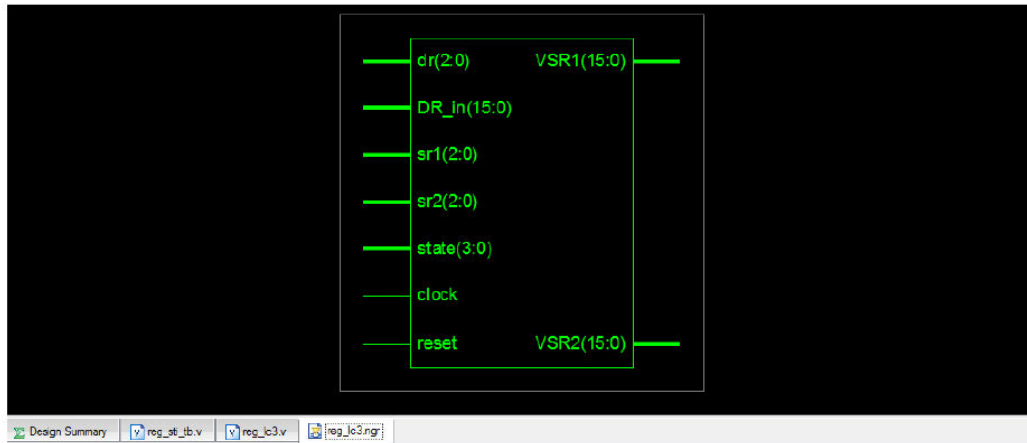


Fig. 3.11 Register File Implementation

3.6.2 Results

Input data :

```

initial
    clock=0;
always
    #5 clock = ~ clock;
initial
    begin
        reset=1;
        #30 reset = 0;
    end

initial
    begin
        state = 4'b1001;
        DR_in = 16'h2457;
        #40 sr1 = 3'b100;
        sr2 = 3'b101;
        dr = 3'b111;
        #15 sr1 = 3'b110;
        sr2 = 3'b111;
        dr = 3'b011;
        #20 sr1 = 3'b011;
        sr2 = 3'b101;
        dr = 3'b000;
    end
end

```

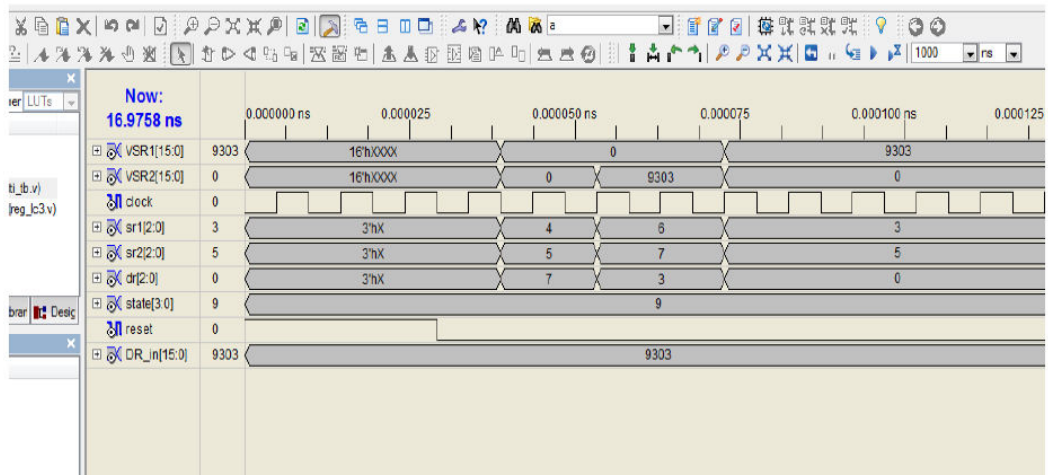


Fig.3.12 Register File Output

3.7 Mem Access Block

3.7.1 Implementation

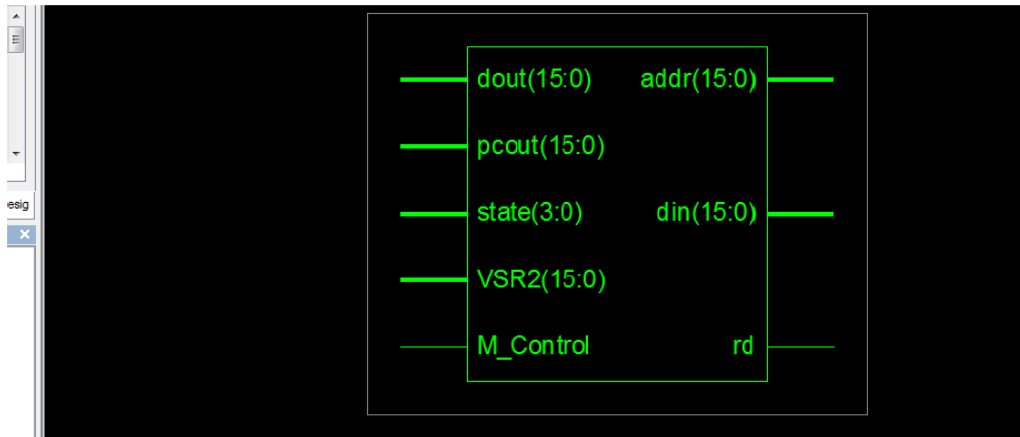


Fig. 3.13 Mem Access Block Implementation

3.7.2 Results

Input data :

```
begin
    state = 4'b0110;
    M_Control = 1'b0;
    pcout = 16'h7846;
    dout = 16'h8648;
    VSR2 = 16'h8536;
    #25 state = 4'b1000;
    M_Control = 1'b1;
end
```

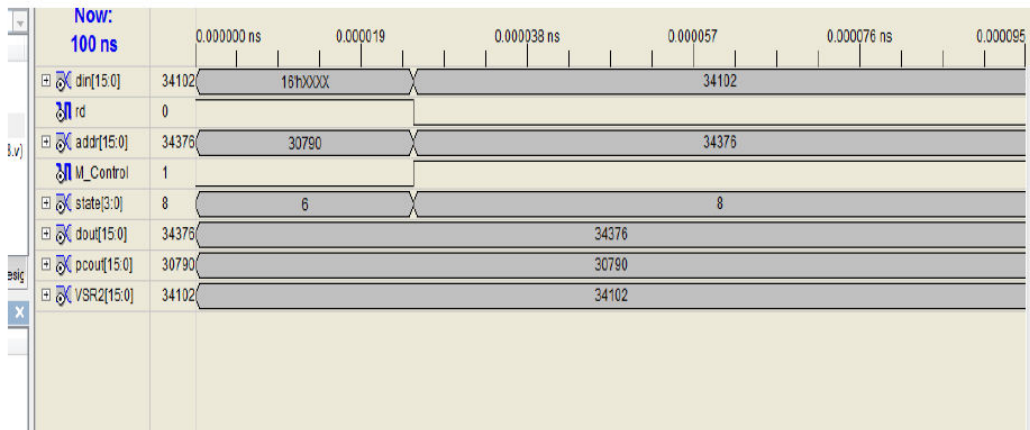


Fig.3.14 MemAccess Block Output

3.8 Top level

3.8.1 Implementation

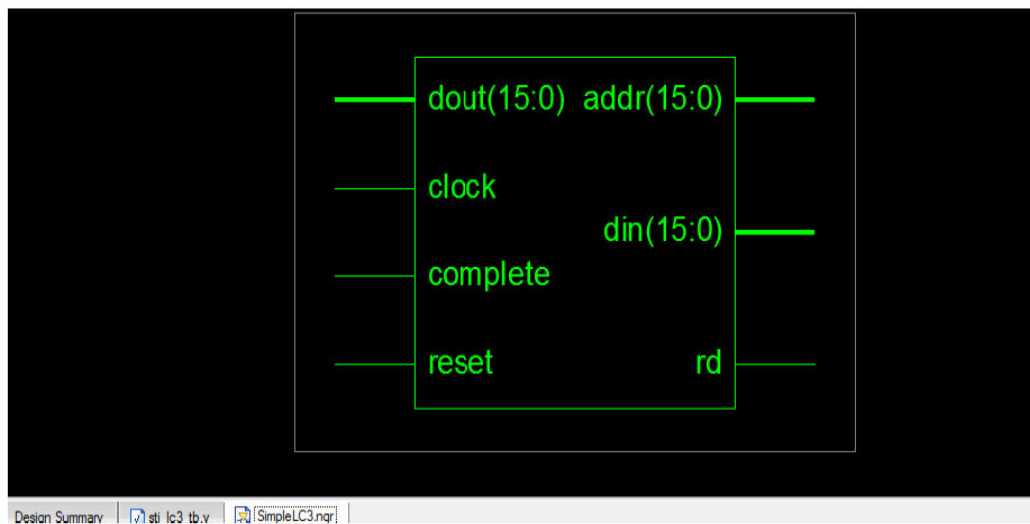


Fig. 3.15 Top level Implementation

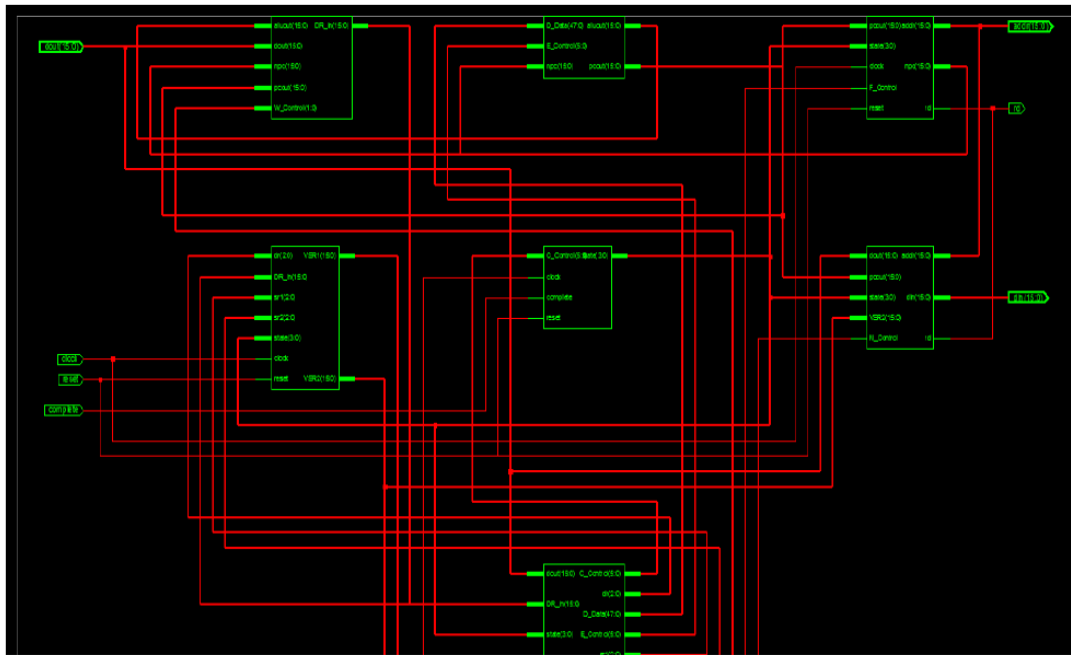


Fig. 3.16 LC3 Schematic Implementation

CONCLUSION

I have successfully implemented the various blocks of LC3 Microcontroller and every instruction completes within 5 clock cycles. I have written the testbenches for each of the blocks of LC3 microcontroller and the results have been verified. I have completed a design involving separate control and datapath with multiple modules and includes most of the elements to be used in the CPU.

FUTURE WORK

In future,i will progress towards a fully pipelined version of LC3 Microcontroller and with a complete set of instructions.

REFERENCES

- [1] Verilog HDL : A guide to Digital Design & Synthesis by Samir Palnitkar
- [2] Verilog Primer by J.Bhasker
- [3] CMOS Digital Integrated Circuits by Sung-Mo Kang and Yusuf Leblebici
- [4] IEEE Standard for Verilog Hardware Description Language
- [5] LC3 Instruction Set Architecture : Department of Electrical and Computer Engineering , NC State University
- [6] <http://elearning.vtu.ac.in/P3/EC74/3.pdf>
- [7] http://conf05.iitkgp.ac.in/avlsi/logf/summercourse/2008/Overview_NBC.pdf