Design and Implementation of 16 bit RISC Processor using Harvard Architecture utilizing minimum area on FPGA

Rajarshi Chattopadhyay[1], Sagnik Raha[2], Rabindranath Sarkar[3], Kaushik Saha[4], Deepan Das[5]

1. Narula Institute of Technology, Kolkata, West Bengal, India. e-mail: likarajo@gmail.com
2. Narula Institute of Technology, Kolkata, West Bengal, India. e-mail: sagnikraha@gmail.com
3. Narula Institute of Technology, Kolkata, West Bengal, India. e-mail: rsrsarkar7@gmail.com
4. Narula Institute of Technology, Kolkata, West Bengal, India. e-mail: kaushik.jis@gmail.com
5. Narula Institute of Technology, Kolkata, West Bengal, India. e-mail: deepandas22@gmail.com

ABSTRACT

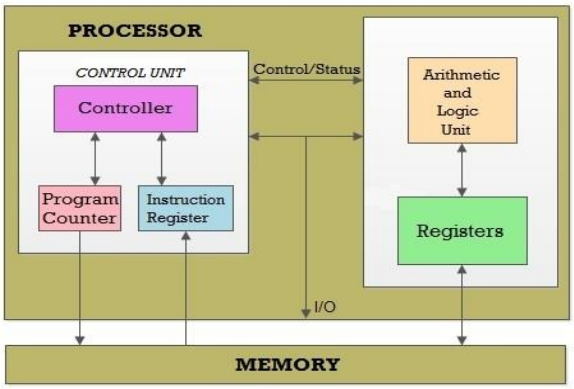
FPGA devices now look like system-on-chips (SoC) with embedded processors, signal processing block, embedded memory interface controller, multi-gigabit transceivers, improved performance and a broad choice of IPs available from FPGA vendors and/or third parties. With this advantage, short design cycle and reconfigurable options, FPGAs are pushing ASICs out of the market. With the increasing popularity of FPGA, optimized architecture design for FPGAs also coming as a challenge to the system designer. No a days, very few Processor IP cores are available for designing a big system on-a-chip. In this project, we have implemented and tested a RISC (Reduced Instruction Set Computing) processor core using FPGA. The CPU is based on Harvard Architecture that consist 2 KB code memory (ROM) and 2 KB data memory (RAM) for program code storage and temporary data storage respectively. It is characterized by 16 bit data bus and 12 bit address bus containing Registers, ALU, and Control. The processor uses in total 14 different instructions including instructions for logical and arithmetical operations. The processor is tested by inserting a program code in code memory. Different test programs have been used to test the working of the processor and corresponding results have been obtained. Both functional and timing simulation fund perfect to justify the working of the processor. This processor utilized only 10K gates in FPGA which is found minimum compared to other processor architecture available in market.

Keywords:
Processor Architecture, RISC, Harvard architecture, 16-bit CPU

1. INTRODUCTION

The development of microprocessor has revolutionized the way we live. Today microprocessors are embedded in nearly every product we use from our cell phone to automobile. Most of the microprocessors are ASIC but FPGA implementation of processor core are rarely available. If a vendor has new features to add at a later stage in the released product, he still has the freedom to decide whether to implement that feature in software or hardware based on applicability. Time-to-market for handling change-requests in FPGA is much less than in ASICs. ASIC may consume more power per unit die size than an FPGA, the power is normally amortised over a higher density solution, which provides better power efficiency. Indeed, this is one of the primary reasons why designers are opting for FPGAs. Designing and implementing an architecture of a processor into a FPGA consist of several constrains like power, speed and area. Our goal is to make a synthesizable code of a 16 bit RISC CPU using VHDL and optimization to meet the desired constrains. A processor usually known as a Central Processing Unit (CPU) is the most essential hardware within a computer system. It is responsible for carrying out the operations based on the instructions of a computer program by performing arithmetical, logical, and input/output operations of the system. A CPU has various important components within it, which work independently or dependently in most cases in a synchronized way to generate a desired output based on an input given to the CPU. The synchronized working is directed by a 'clock' signal which the CPU receives as an input signal.

1.1 Architectural Overview of generic processor:



The components of a generic processor are:-

- *Control unit (CU)*: It is responsible for supervising all the operations of the system. It direct the other parts of the entire computer system to execute stored program instructions communicating with both the ALU and memory.
- *Arithmetic and Logic unit* (ALU): It is a digital circuit that provides logical and computational capabilities to the processor.
- *Register*: It is the storage location inside the processor addressed by typical mechanisms and can be accessed quickly. Data can be used for operations or loaded from memory in it.
- *Program Counter*: It holds the memory address of the next instruction that would be executed by the processor. Content of PC increments after each instruction cycle is complete. The content of the PC can also be changed by certain instructions.
- *Instruction Register*: It stores the instruction code currently being executed.

1.2 RISC Vs CISC Processor:

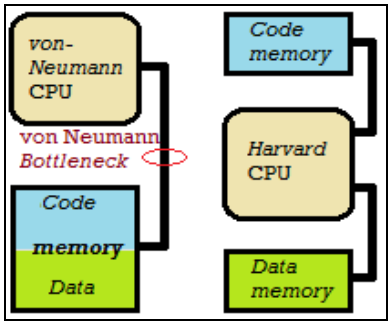
Reduced Instruction Set Computing, or RISC, is a CPU design strategy based on the insight that simplified instructions can provide higher performance with faster execution of each instruction. RISC uses a small, highly-optimized set of instructions, rather than a more specialized set of instructions.

Complex Instruction Set Computing, or CISC, is a CPU design strategy where single instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) or are capable of multi-step operations or addressing modes within single instructions.
For completing an operation there are more lines of code and so more memory is needed to store instructions in RISC. More work has to be done by compiler to convert a high-level language statement into code of this form. However, each instruction requires only one clock cycle to execute, thus entire program will execute in approximately the same amount of time as a single CISC command. RISC instructions require less hardware space leaving more room for registers. In RISC all of the instructions execute in a uniform amount of time (i.e. one clock cycle), so pipelining is possible. After a CISC-style command is executed, the processor automatically erases the registers. In RISC, the operand will remain in the register until another value is loaded in its place.

1.3 Harvard and Von Neumann Architecture:

The Von Neumann architecture has the same memory block for code and data, i.e. instructions and data are stored in the same memory subsystem and share a communication pathway or bus to the CPU. An instruction fetch and data operation cannot be performed at the same time. This constraint is referred to as the *Von Neumann bottleneck* and limits the throughput or data transfer rate between the CPU and memory

Harvard architecture has separate data and code memory with separate data and instruction pathways allowing instruction and data to be accessed concurrently which allows Pipelining - fetch while executing another. This decreases the execution time of one instruction.



2. 16 BIT RISC PROCESSOR ARCHITECTURE DESIGN

2.1 Motivating background

The study of INTEL 8085 microprocessor and practicing the use of it during graduation enhanced our interest in CPU designing. 8085 itself being a processor based on the von Neumann architecture, using the same memory space (RAM) for instruction code and data storage influenced us to design our own processor using the Harvard architecture scheme. Interest to use simple instructions for programs made us follow RISC strategy, simplifying our CPU design.

2.2 Processor Description

The 16-bit RISC CPU designed is capable of addressing up to 4K bytes (4096 bytes) of memory.
It is based on Harvard architecture with 2K bytes code memory ROM and 2K bytes data memory RAM.
RISC approach has been taken here to simplify instructions and make execution faster.

Processor components:

- Arithmetic and logic Unit (ALU)
- Accumulator (Acc) - 16 bit register

- Program Counter (PC) - 12 bit register
- Instruction Register (IR) - 4 bit register
- Memory Address Register (MAR) - 12 bit register
- Control unit (CU)

Bus:
- Data bus (16 bit)
- Address bus (12 bit)

Memory:
- Program memory (ROM) - 2K bytes)
- Data Memory (RAM) – 2K bytes)

Operation Summary: The PC gives out 12 bit address of the instruction code to be fetched through the outgoing Address Bus port connected to the memory. 16 bit data containing instruction incoming from the memory gets distributed into 4 bit IR and 12 bit MAR. The 4 bits of the data D_{15}-D_{12} form the operation code and move to IR, and the 12 bits D_{11}-D_0 giving the memory address of the data which has to be used for operation move to MAR. Depending on the operation code, either they are fed to the ALU for operation or the CU decides how the different components within the CPU handle the completion of the operation. A data can be loaded to the Acc from the specific memory location corresponding to that particular data. The arithmetic and logic operations are done between the data previously loaded in the Acc and another data stored in the memory, pointed to by the address in MAR. The result of such operation overwrites the data in the ALU, which can then be stored to any RAM location. The non-arithmetic and logic operations are supervised by the CU.

2.3 Processor Component Functions

Arithmetic Logic Unit: For arithmetic and logical operations.

Accumulator: The register in which data is operated and manipulated.

Program Counter: The register which contains the address of the memory location of the next instruction. It gets updated after fetching an instruction or on encountering JUMP statements.

Instruction Register: It contains the op-code of the instruction that has been fetched by the processor.

Memory Address Register: The MAR contains the address portion (12-bit) of the instruction. It also has access to the PC.

Data Bus: The Data Bus is a set of 16 lines and they are bidirectional as data can flow out of as well as into the CPU.

Address Bus: The Address Bus is a set of 12 lines and is used for Read/Write operations. Also, the address lines can be used to selectively turn ON specific integrated circuit devices; these lines then act as *chip selects* or *chip enables.*

Program Memory: This contains the program code to be executed.

Data memory:This can be accessed during data storing operation.

Control Unit: The Control Circuitry is responsible for all the operations. The Control Circuitry and all operations are driven by the clock signals. All operations are synchronously timed according to the clock signals. Control Lines such as Read (RD), Write (WR) lines originate from this unit. A high on RD indicates that data on the Data Bus which has been placed on it by the selected memory is to be read into the processor. A high on WR indicates that data on the Data Bus which has been placed on it by the processor is to be written into the selected memory location.

2.4 Instruction Set

Operation code is of 4 bits, which implies that in total 2^4 i.e. 16 instructions can be made available for this processor. However, only 14 instructions are there for use in a program for this processor.
The different instructions are briefly stated along with their operation code in the following table:

Op-Code	Instruction	Operation
0h	ADD #Addr	A ← [A] + [#Addr]
1h	SUB #Addr	A ← [A] - [#Addr]
2h	MUL #Addr	A ← [A] * [#Addr]
3h	DIV #Addr	A ← [A] / [#Addr]
4h	AND #Addr	A ← [A] AND [#Addr]
5h	OR #Addr	A ← [A] OR [#Addr]
6h	XOR #Addr	A ← [A] XOR [#Addr]
7h	NOTA	A ← NOT [A]
8h	LDA #Addr	A ← [#Addr]
9h	STA #Addr	#Addr ← [A]
Ah	JZ #Addr	PC ← #Addr if [A] = 0
Bh	JNZ #Addr	PC ← #Addr if [A] not = 0
Ch	JMP #Addr	PC ← #Addr
Dh		
Eh		
Fh	END	PC ← 000

Note:

#Addr => Hexadecimal value of address (000 to FFF)
[#Addr] => The content of memory address #Addr
A => Accumulator register
[A] => content of the Accumulator

- *ARITHMETIC GROUP*
  ADD #Addr, SUB #Addr,MUL #Addr, DIV #Addr
- *LOGIC GROUP*
  AND #Addr, OR #Addr, XOR #Addr, NOT Addr
- *DATA TRANSFER GROUP*

LDA #Addr, STA #Addr

- *BRANCH GROUP*

  JZ #Addr, JNZ #Addr, JMP #Addr

- OTHER

  END

Example of code fed to the processor:

- 2400h => multiply (op-code '2h') the content of the Acc with the content of memory location '400h' and store the result in the Acc.
- 8780h => Load (op-code '8h') the content of memory location '780h' into the Acc.
- 9850h => Store (op-code '9h') the content of Acc to the memory location '850h'.
- C200h => Jump (op-code 'Ch') to the memory location '200h'.

2.5 ALU MODELLING

The Arithmetic Logic Unit (ALU) in the microprocessor is responsible for carrying out all kind of arithmetic operations such as and different logical.

The ALU designed as a part of the CPU, has the following blocks:

- *Arithmetic Block*

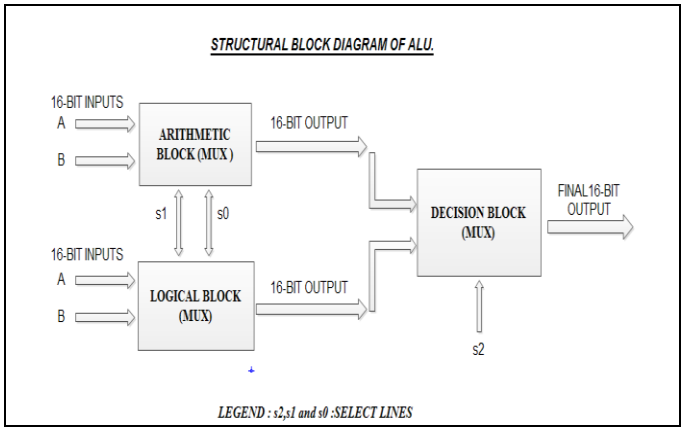
There are two select lines with which we can choose any one of the operations - ADD, SUB, MUL, DIV. Addition, Subtraction and Multiplication can been realized with VHDL operators +, - , and * respectively which are synthesizable. In case of Division the '/' operator is not synthesizable. Hence, a separate Divider following ***Non-Restoring Division algorithm*** is designed and included in the ALU as a subcomponent of the Arithmetic Block.

- *Logical Block*

The Logical Block also has two select lines to choose any one of the four logical operations - AND, OR, XOR, NOT.

- *Decision Block*

This block has a *single select line* which indicates whether the result of the Arithmetic Block or the Logical Block is to be considered as the final result.



Working of the ALU:

Inputs: 16-bit data input (A and B) and Select Lines (s2, s1 and s0).

The data input is common to both the Arithmetic Block and the Logical Block. Select lines s1 and s0 are common to Arithmetic Block and the Logical Block. Select line s2 is connected to the Decision Block.

The select lines s1 and s0 select the operation Addition, subtraction, multiplication and division in AU Block and AND, OR, XOR and NOT in Logical Block.

For choosing whether the result of the Arithmetic Block or the Logical Block is to be considered as the final result the select input s2 is responsible.

The different values for the Select inputs and their corresponding operations are listed as follows:

s2	s1	s0	Operation
0	0	0	ADD
0	0	1	SUBTRACT
0	1	0	MULTIPLY
0	1	1	DIVIDE
1	0	0	AND
1	0	1	OR
1	1	0	XOR
1	1	1	NOT

2.6 MEMORY MODELLING

ROM Modeling:

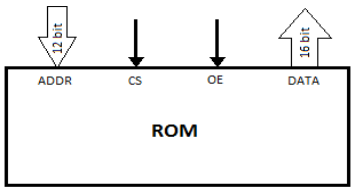
A *ROM (Read Only Memory)* is a memory block/device in which data is put in different memory locations during its design stage, which cannot be overwritten by the processor at any time. It acts as a *non volatile storage* space of data. Thus, only the data can be read from it during any program by the processor.

Design: The ROM designed for the processor is a memory block of an array of 2048 (for 2K bytes) memory locations.

Inputs: Chip Select (CS), OE (Output Enable), Address (ADDR) 12bit

Output: Data (DATA) 16 bit

When CS is '1' (high), then it gets selected. For the processor to read data from the ROM, a high signal has to be given to the *OE*. The address of the memory location from which data is to be read has to be given as the input through *ADDR*. The data present in the particular memory address received as input, is given as the output through the *DATA port.*



RAM Modeling:

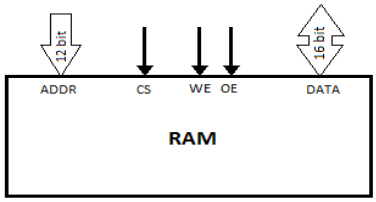
A *RAM (Random Access Memory)* is a memory block/device which allows data items to be read from and written to any memory location randomly i.e. at anytime as desired, by the processor. It acts as a *volatile storage* space of data. Thus data can be both read from and written to it during any program by the processor.

Design: The RAM designed for the processor is a memory block of an array of 2048 (for 2K bytes) memory locations.

Inputs: Chip Select (CS), OE (Output Enable), WE (Write Enable), Address (ADDR) 12 bit;

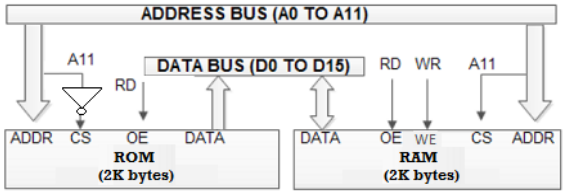
Bidirectional: Data (DATA)16 bit

When CS is '1' (high), then it gets selected. For the processor to read data from the RAM, a high signal has to be given to the *OE*. The address of that memory location from which data is to be read has to be given as the input to the RAM through *ADDR port*. The RAM gives the data present in the particular memory address received as input, as the output through the *DATA port.* For the processor to write data to the RAM, a high signal has to be given to W*E*. The data to be written has to be given as input to the RAM through *DATA port*, which is written in the particular memory address of the RAM received as input through the *ADDR port.*



Memory interfacing of ROM and RAM:

The *12 bit Address Bus (A_0 -A_{11})* is connected to the ROM and the RAM through their respective *12 bit ADDR port*.
The *16 bit Data Bus (D_0-D_{15})* is connected to the ROM and the RAM through their respective *16 bit DATA port*. The line A_{11} is used as the *chip select* input for the ROM and the RAM. It is connected via a *NOT gate* to the *CS* input of the ROM and is connected directly to the *CS* input of the RAM. When A_{11} is '0' (low), then ROM gets selected and when A_{11} is '1' (high), then RAM gets selected.
So, ROM address ranges from 000h to 7FFh, and RAM address ranges from 800h to FFFh, each of memory 2K bytes. The READ (*RD*) signal from the processor is an input to both ROM and RAM. RD signal line is connected to the *OE* (Output Enable) of both the ROM and the RAM. The WRITE (WR) signal from the processor is an input to RAM. WR signal line is connected to the *WE* (Write Enable) of the RAM.



2.7 Control Unit Modeling

CU Functions:

- *Phase generation:*
  At the beginning the phase is by default '0' i.e. low.
  For every rising edge of the clock, the phase changes to its opposite nature, i.e. low to high ('0' to '1') or vice versa.
  Phase = '0' => Op-code fetch cycle
  Phase = '1' => Execution cycle

- *READ (RD) signal generation*
  In Op-code fetch cycle, and for every instruction in Execution cycle other than the STA, as detected from the 4 bit code in IR, the output RD line of the processor is made high (i.e. '1'). For other cases it is low (i.e. '0').

- *WRITE (WR) signal generation*
  For only the STA instruction in Execution cycle, the output WR line of the processor is made high (i.e. '1'). For other cases it is low (i.e. '0').

- *Data into ADDRESS BUS*
  In Op-code fetch cycle, data in PC is put into the Address Bus and in Execution cycle, data in MAR is put into the Address Bus, via ADDR port of CPU.

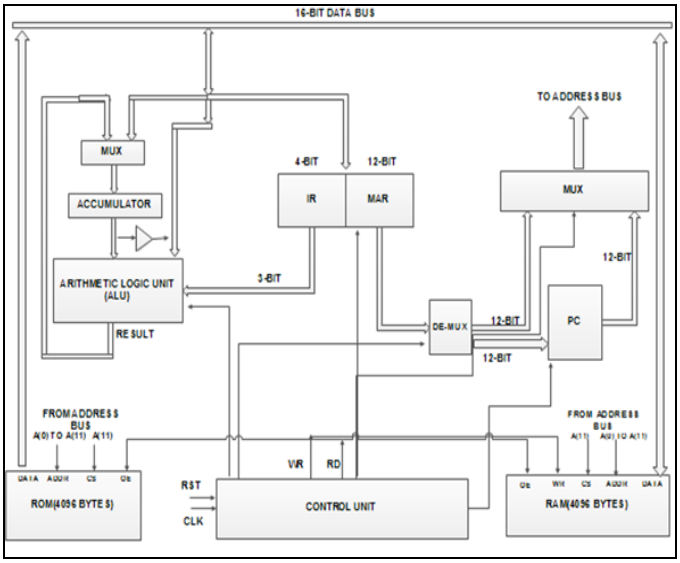
- *Data into DATA BUS*
  In Execution cycle, only for STA instruction, the result of the Accumulator (ACC) is put into the Data Bus via DATA port of CPU. For other case it is kept at High Impedance state.

- *PROGRAM COUNTER (PC) update*
  If RESET signal is detected to be high (i.e. '1'), the PC is reset to ZERO (12 bits '0'). If RESET signal is low, then for every rising edge of clock signal, the content in PC is incremented by ONE.
  In Execution Cycle, for JNZ instruction, PC is loaded with content of MAR if Acc content is NOT ZERO. For JZ instruction, PC is loaded with content of MAR if Acc content is ZERO. For JMP instruction, PC is loaded with content of MAR. For END instruction, PC is forcefully set to ZERO (12 bits '0').

- *Op-code and Operand Fetch*
  When rising edge of clock is detected, then if the phase indicates Op-code fetch cycle, the contents of the Data Bus are read via DATA port. The higher 4 bits of data (D_{15}-D_{12}) are sent to the IR and the remaining 12 bits (D_{11}-D_0) are sent to the MAR. The contents of the IR give the Op-code and the contents of the MAR give the Operand.

- *ACCUMULATOR (Acc) update*
  When rising edge of clock is detected, then if the phase indicates Execution cycle, and the instruction is an *Arithmetic group* or *Logic group* instruction, then the Acc is loaded with the result of the ALU. If LDA instruction is encountered, then the Acc is loaded with the data from the Data Bus coming through the DATA port of the CPU.

2.8 Overall CPU Integration



3. EXPERIMENTS AND RESULTS

3.1 Hardware Platform for Implementation

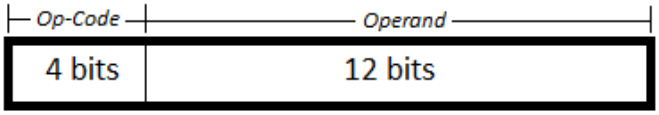
VHDL (VHSIC Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits.

Xilinx ISE (Integrated Software Environment) is a software tool produced by *Xilinx* for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. This has been used here to write the VHDL program for the processor, memory and display unit.

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. Here, *Spartan 6 FPGA* kit has been used to implement the processor and its functionality along with integrated memory and display unit interface for testing purpose.

3.2 Problem Descriptions and Simulation Results

The program code consists of instructions of 16 bit, which are placed one by one in ROM locations starting from 000h. The PC of the processor accesses the ROM for the Op-code starting from location 000h till the END instruction is encountered.



e.g. for instruction 5678h, 5h is the Op-code and 678h is the operand.

NOTE: In the ROM memory, values 0 to 9 are permanently loaded in locations 700h to 709h respectively. Program code starts from location 000h.

TEST OPERATION 1:

Algorithm Implemented

((1+2) x (4/2)) – 2;

Expected Output: 4

TEST CODE:

LDA 701 - load value ‘1’ at ROM loc 701h to Acc
ADD 702 - add value ‘2’ at ROM loc 702h to content of Acc
STA 800 - store result in Acc to RAM loc 800h
LDA 704 - load value ‘4’ at ROM loc 704h to Acc
DIV 702 - divide content of Acc by value ‘2’
STA 801 - store result in Acc to RAM loc 801h
LDA 800 - load value in RAM loc 800h to Acc
MUL 801- multiply content of Acc by value at RAM loc 801h
STA 802 - store result in Acc to RAM loc 802h
LDA 802 - load value in RAM loc 802h to Acc
SUB 702 - subtract value ‘2’ at ROM loc 702h from Acc content
STA 803 - store result in Acc to RAM loc 803h
END - jump to ROM loc 000h i.e. start of program

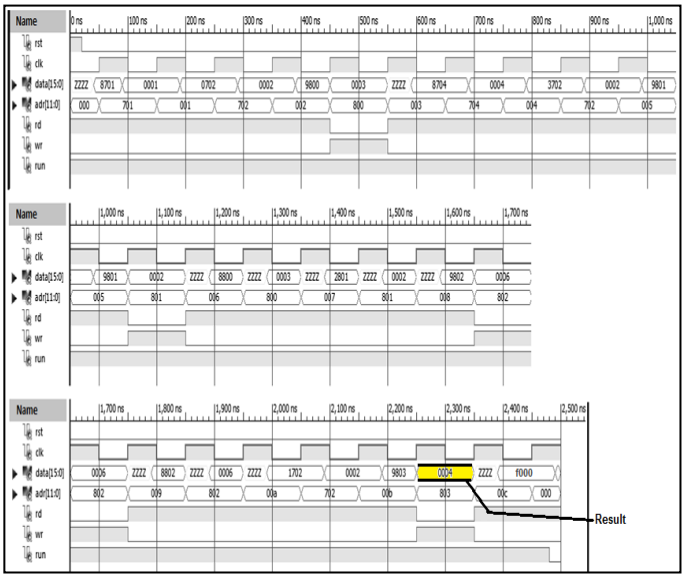
The hex code to be written in ROM from 000h is as follows:

8701, 0702, 9800, 8704, 3702, 9801, 8800, 2801, 9802, 8802, 1702, 9803, F000

Note: The underlined portion can take any value, since the first 'F' is enough for processor to know it's a JUMP to 000h command

FUNCTIONAL SIMULATION RESULT:

Final output - yellow region



TEST OPERATION 2:

Algorithm Implemented

(2 OR 7) AND (6 XOR 9);

Expected Output: 7

TEST CODE:

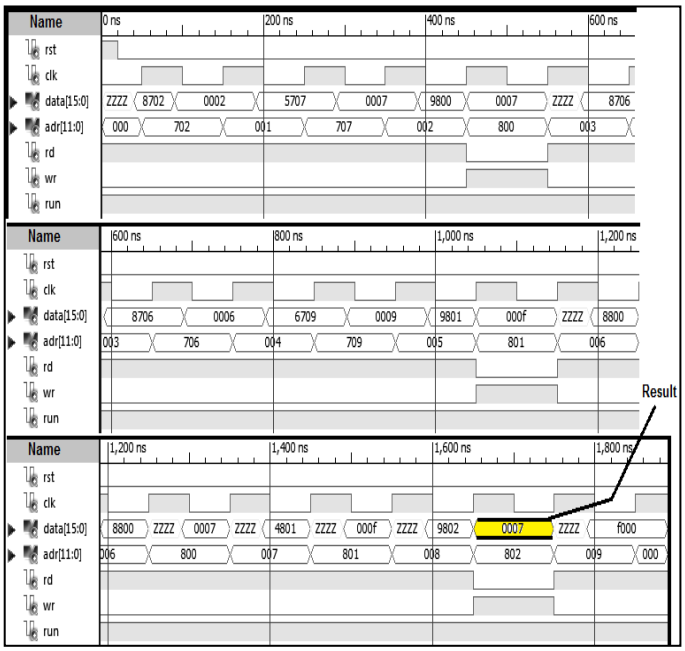
LDA 702 - load value '2' at ROM loc 702h to Acc
OR 707 - OR value '7' at ROM loc 707h to content of Acc
STA 800 - store result in Acc to RAM loc 800h
LDA 706 - load value '6' at ROM loc 706h to Acc
XOR 709 - XOR value '9' at ROM loc 709h to content of Acc
STA 801 - store result in Acc to RAM loc 801h
LDA 800 - load value in RAM loc 800h to Acc
AND 801- AND content of Acc by value at RAM loc 801h
STA 802 - store result in Acc to RAM loc 802h
END - jump to ROM loc 000h i.e. start of program

The hex code to be written in ROM from 000h is as follows:

8702, 5707, 9800, 8706, 6709, 9801, 8800, 4801, 9802, F000

FUNCTIONAL SIMULATION RESULT:

Final output- yellow region



Synthesis Result and Performance Comparison:

CPU Project Status (04/28/2014 - 01:05:19)			
Project File:	CPU16_TB.xise	**Parser Errors:**	No Errors
Module Name:	CPU	**Implementation State:**	Synthesized
Target Device:	xc3s100e-4tq144	**• Errors:**	No Errors
Product Version:	ISE 14.6	**• Warnings:**	2 Warnings (2 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	269	960	28%
Number of Slice Flip Flops	45	1920	2%
Number of 4 input LUTs	513	1920	26%
Number of bonded IOBs	32	108	29%
Number of MULT18X18SIOs	1	4	25%
Number of GCLKs	1	24	4%

Processor	No. of Instructions	No. of Gates
8085 (CISC)	246	6500
80286 (RISC)	20,000	1,34,000
Our Processor (RISC)	14	10,000 (including memories)

4. CONCLUSION AND FUTURE RESEARCH

4.1 Conclusion

The simulation results show that the result obtained matches with the desired theoretical value which justifies the proper functionality and working of the designed CPU and performance achieved by this design is similar to other ASIC processor based on Harvard architecture.

4.2 Future Research

The CPU working on 16 bit data line and 12 bit address line is totally modifiable. This processor based on Harvard architecture and our future target is to include more instructions and pipeline architecture.

ACKNOWLEDGMENT

We are greatly thankful to Mr. Sohan Ghorai, Assistant Professor of Dept. of Electronics & Communication Engineering, Narula Institute of Technology, for his guidance and support throughout this project and making it successful one.

REFERENCES

1. Volnei A. Pedroni, “Circuit design with VHDL”, Elsevier, 2013
2. M. Morris Mano, “Digital Logic And Computer Design”, 2nd Edition, 2002
3. David.M. Harris, Sarah.L. Harris, “Digital Design and Computer Architecture”, 2nd Edition, Elsevier, 2013
4. Wayne Wolf, “FPGA based System Design”, 1st Edition, PHI, 2005
5. http://en.wikipedia.org
6. http://www.xilinx.com/products/silicon-devices/fpga/spartan-6/index.htm

Rajarshi Chattopadhyay pursuing B.Tech in Electronics and Communication Engineering, from Narula Institute of Technology, Kolkata, West Bengal, India. His technical interest includes FPGA, Microprocessor and Microcontroller, Digital Design and Software Design.

Sagnik Raha pursuing B.Tech in Electronics and Communication Engineering, from Narula Institute of Technology, Kolkata, West Bengal, India. His technical interest includes FPGA, Microprocessor and Microcontroller, Digital Design and Software Design.

Rabindranath pursuing B.Tech in Electronics and Communication Engineering, from Narula Institute of Technology, Kolkata, West Bengal, India. His technical interest includes FPGA, Microprocessor and Microcontroller, Digital Design and Software Design.

Kaushik Saha pursuing B.Tech in Electronics and Communication Engineering, from Narula Institute of Technology, Kolkata, West Bengal, India. His technical interest includes FPGA, Microprocessor and Microcontroller, Digital Design and Software Design.

Deepan Das pursuing B.Tech in Electronics and Communication Engineering, from Narula Institute of Technology, Kolkata, West Bengal, India. His technical interest includes FPGA, Microprocessor and Microcontroller, Digital Design and Software Design.