“Design And Implementation Of RISC Processor”

Submitted in partial fulfillment of the requirements
of the degree of Bachelor of Engineering

By

Names	Roll No.(DIV-D11)
Vansh Dhoka	9
Sukanya Pingle	34
Atharva Godkar	50
Anish Godse	51

Supervisor:
Dr Jaymala Adsul

Department of Electronics Engineering

V.E.S. Institute of Technology

(Autonomous Institute Affiliated to University of Mumbai, Approved by AICTE & Recognised by Govt. of Maharashtra)

2023-24

CERTIFICATE

This is to certify that the project entitled “Design And Implementation Of RISC Processor” is a bonafide work of Vansh Dhoka, Sukanya Pingle, Atharva Godkar, Anish Godse Submitted to the V.E.S. Institute of Technology in partial fulfillment of the requirement for the award of the Bachelor of Engineering in **Electronics Engineering.**

(Name and sign)	(Name and sign)
Supervisor/Guide	Co-Supervisor/Guide
(Name and sign)	(Name and sign)
Head of Department	Principal

Project Report Approval for T.E

This project report entitled **"**Design And Implementation Of RISC Processor**"** by Vansh Dhoka, Sukanya Pingle, Atharva Godkar, Anish Godse is approved for the degree of the Bachelor of Engineering in **Electronics Engineering.**

Examiners

1. _____

2. _____

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Name and Signature of Students	Name and Signature of Students
Name and Signature of Students	Name and Signature of Students

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

INTRODUCTION

Aim of the project –

The aim of our project is to design a RISC processor capable of executing simple programs written in assembly language. We will be making a 32 bit RISC processor. All registers will be of 32 bit. There will be 2Kb of RAM and 2Kb of ROM. A register bank having 32 general purpose registers. All data BUS will have 32 bit width. Whereas the width of address bus will keep varrying. It should be able to perform all sorts of logical and mathematical operations required to obtain the result.

This processor can be used to run the following programs –

Arranging data in ascending order

Arranging data in descending order

Calculating HCF & LCM of a number

Calculating the factorial of a number

Finding prime & composite numbers out of given data

Finding odd & even values in a given data

Finding max & min values in a given data

Work flow –

For designing this processor we will be using VIVADO software. First of all we will write verilog code for each and every module separately and test it. Finally we will join all these modules under the top level module. For the testing purpose we will be using the simulation tool available on vivado software. Later we will test the same module on FPGA. We will write a code to interface FPGA to keyboard. This will help ous to write assembly level codes directly through keyboard. Also we will right a code for interfacing PC monitor to FPGA. This will make it easy for ous to see the 32 bit output of the processor on the screen. For keyboard interfacing we will be making use of the PS2 protocol. For monitor interfacing we will use VGA protocol. First will be going for single cycle processor. If it works fine, then we shall be introducing pipeling to our design. We aim to add 5 stage pipeline, which includes – fetch, decode, execute, memory, write back.

Problem statement & proposed solution –

We have designed the simplest possible RISC processor. This is the uniqueness of our project. While designing a processor lot of parameter come into picture. stages of pipelining, number of cores, heat dissipation, on chip area, computational speed etc. Engineers all over the world work on these factor is to boost the efficiency of processor. Our project specifically deals with the design simplicity part of RISC processor. Maximum number of operation should be executed with minimum number of instructions types. Also the BUS architechture and memory organisation is done in as simple way as possible.

CHAPTER 2
LITERATURE REVIEW

LITERATURE REVIEW

Compared with hardcore processors, adding softcore processors can help FPGA to improve reliability. Many existing soft processors only aim at minimizing FPGA resources consumption or achieving high performance. However, a high-performance processor with low-resource consumption is demanded in implementing the hardware accelerator. To achieve this goal, a 32-bit soft processor based on the RISC-V instruction is proposed.
(T. Zheng, G. Cai and Z. Huang, "A Soft RISC-V Processor IP with High-performance and Low-resource consumption for FPGA," 2022 IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, 2022, pp. 2538-2541, doi: 10.1109/ISCAS48785.2022.9937742.)

RISC-V stands for 'reduced instruction set computer (RISC) five'. The number five refers to the number of generations of RISC architecture that were developed at the University of California, Berkeley since 1981. The RISC concept was motivated by the fact that most processor instructions were not used by most computer programs. Thus, unnecessary decoding logic was being used in processor designs, consuming unnecessary power and silicon area. The alternative was to simplify the instruction set and to invest more in register resources.
(Urquhart, Roddy (29 March 2021). "What Does RISC-V Stand For? A brief history of the open ISA". *Systems & Design: Opinion*. Semiconductor Engineering.)

The integrated circuit has come a long way since Jack Kilby's first prototype. His idea founded a new industry and is the key element behind our computerized society. Today the most advanced circuits contain several hundred millions of components on an area no larger than a fingernail. The transistors on these chips are around 90 nm, that is 0.00009 millimeters*, which means that you could fit hundreds of these transistors inside a red blood cell. ("The History of the Integrated Circuit". Nobelprize.org. Archived from the original on 29 Jun 2018. Retrieved 21 Apr 2012.)

CPU cache memory is divided into an instruction cache and a data cache. Harvard architecture is used as the CPU accesses the cache. While the CPU executes from cache, it acts as a pure Harvard machine. When accessing backing memory, it acts like a von Neumann machine. This technique is used in some microcontrollers, including the Atmel AVR. This allows constant data, to be accessed without first having to be copied into data memory, preserving scarce data memory for read/write variables. Special machine language instructions or peripheral interference is used to do the same.
(Pawson, Richard (30 September 2022). "The Myth of the Harvard Architecture". *IEEE Annals of the History of Computing*. **44** (3): 59–69. doi:10.1109/MAHC.2022.3175612. S2CID 252018052)

The 32 bit RISC-V processor comprises of load and store architecture. It means having two instructions for accessing memory LOAD instruction is to load the data from the memory and STORE instruction is used to write the data in to the memory and also uses the Hardwired architecture to increase the performance of the processor. "Design Of 32 Bit Asynchronous RISC-V Processor Using Verilog", International Journal of Emerging Technologies and Innovative Research (www.jetir.org | UGC and issn Approved), ISSN:2349-5162, Vol.7, Issue 3, page no. pp1717-1722, March-2020,

This paper briefs on the 32-bit, 5-stage superscalar architecture. The processor supports the pipelining feature which increases the performance of the architecture. 5- stages of pipeline are IF, ID, EX, MEM and WB stage. This work particularly includes the implementation on sub-blocks of the processor like Floating point unit, Branch prediction unit, forwarding unit, operand logic and floating point register file presented in different stages of the pipelined processor. Implementation of these blocks is done using Verilog HDL and is simulated in Xilinx ISE. International Research Journal of Engineering and Technology (IRJET) Volume: 08 Issue: 06 | June 2021 . Shruthi1, Dr. Jamuna S2 1PG Student (M.Tech in VLSI Design and Embedded Systems), Department. of ECE, DSCE Bangalore, Karnataka 2Professor, Department. of ECE, DSCE, Bangalore, Karnataka

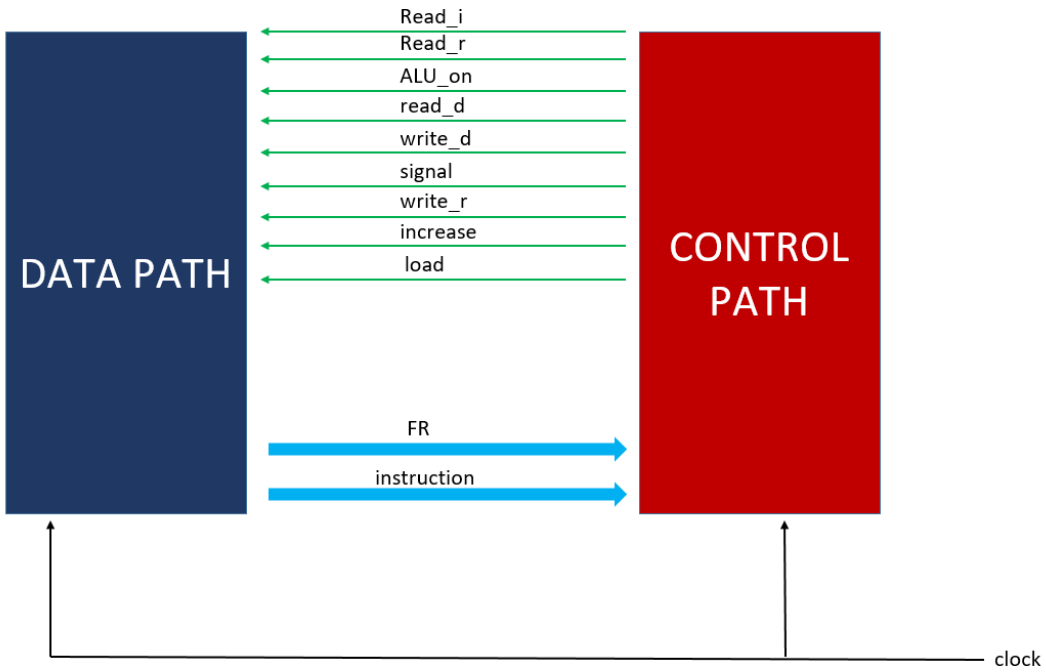
The processor is based on Harvard architecture. The instruction set adopted here is extremely simple that gives an insight into the kind of hardware which should be able to execute the set of instructions properly. Along with sequential and combinational building blocks of a processor such as adders and registers more complex blocks such as ALU and memories have been designed and simulated. The modelling of ALU which has been done in this project is fully structural starting from half adders. The Reduced Instruction Set Computer, or RISC, is one whose instruction set architecture allows it to have fewer cycles per instruction. Mr. Rajkumar D. Komati1, Aditya Kolekar2, Kunal Kasbekar3, Ms. Avanti Godbole4 1Assistant Professor, Dept. of ENTC, MIT College of Engineering, Kothrud, Pune 2, 3, 4Undergraduate Student Dept. of ENTC, MIT College of Engineering, Kothrud, Pune.

In a pipelined RISC MIPS processor, the data unit is responsible for performing memory operations, such as loading and storing data to and from memory. The data unit interfaces with the memory subsystem of the processor to perform these operations. In a pipelined RISC MIPS processor, the instructions and decode unit is responsible for fetching and decoding instruction from memory. o values. Data movement operations involve moving data between different memory locations or between the processor’s registers. The implementation of RISC MIPS processor has a set of registers, which are small memory locations used for storing data temporarily. The instructions are loaded from memory into the processor, and then executed by the different stages of the pipeline. International Journal for Research in Applied Science & Engineering Technology (IJRASET) ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538 Volume 11 Issue IV Apr 2023- Available at www.ijraset.com Authors: Manjula B B, Vidyashree H, Kavyashree M G, Tulasi V, Arpitha R

CHAPTER 3

BLOCK DIAGRAM & WORKING

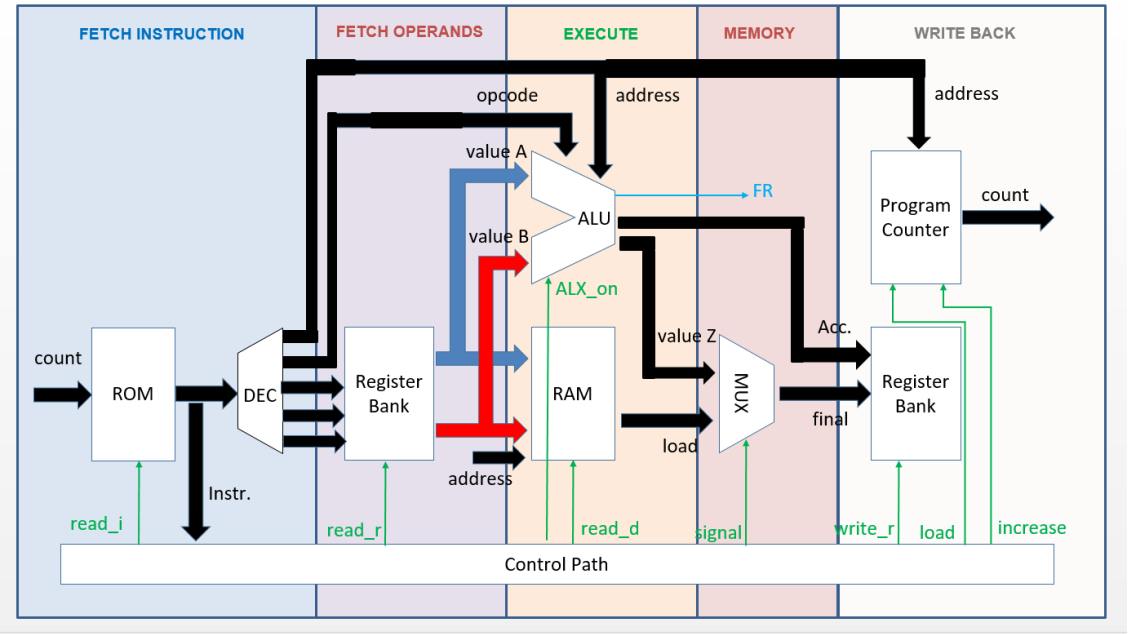
BLOCK DIAGRAM & WORKING



The entire project consists of three main verilog codes – control path, data path, and testbench. The data path is a top level module and consists of various sub modules under it. This method is called as instantiation in verilog. The data path mainly deals with how the data is being fetched, processed and stored back into memory. All the digital electronic circuits like mux, demux, flip flops, PIPO registers, memory, encoders, decoders etc come under data path. These circuits are connected to one another via BUS. Another important job of data path is to generate status signals (flags). These signals are sent to control path, depending on which appropriate control signals are generated.

The control path deals with generating of control signals. The execution of instruction is broken down into 5 stages. In other words execution of instruction takes 5 T states. (please note that we are not talking 5 stages of pipeling over here). During each stage some of the circuits work to produce output while rest of the circuit remain off. To decide all this we need to give proper control signal (enable) to each and every circuit. For each of the 5 stages unique control signals are generated. The current stage of control path keeps rotating from 1 to 5 and back to 1. This marks the completion of one instruction. The shift from current stage to next stage is determined by FSM. FSM stands for finite state machine. It is further classified into 2 types – mealey machine and moore machine. Our control path is basically hardware implementation of moore machine. Incase of a moore machine the next stage depends only on current stage and not current input.

Finally we will talk about the test bench. In verilog a testbench is written to verify the logic of the code. Different inputs are applied to the DUT at different time intervals. The output produed by DUT is recorded in the form of a timing diagram. Study of this timing diagram tells ous whether the verilog code is correct or not. DUT stands for design under test. It is basically a copy of module that is created under testbench. The output of testbench can even be observed in TCL console of the vivado software.

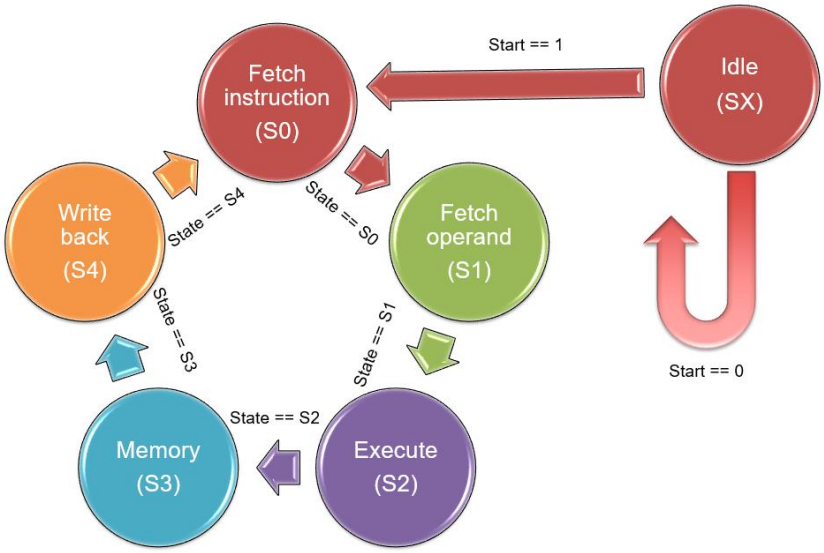


Design specidifications of RISC processor

- 2Kb ROM, so 11 bit address BUS is required 2^11 = 2048 = 2Kb
- 2Kb RAM, so 11 bit address BUS is required 2^11 = 2048 = 2Kb
- 1:8 decoder is used. input is 32 bit instruction. 3/8 outputs are undefined.
- Output is address of registers (5x3 bit), opcode(6 bit) and immediate value(11 bit)
- A register bank having 32 general purpose registers. 5 bit address BUS 2^5 = 32
- Total 23 instructions, so 5 bit opcode required 2^5 = 32 > 23.
- However we purposely took 6 bit opcode incase we wish to add more instruction in future.
- Program counter (synchronised up counter) is of 11 bits.
- 32 bit flag register to hold status flag. 28/32 flags are undefined.

Working of data path–

PC sends the address of next instruction to ROM. The instruction from ROM is sent to decoder. At the decoder the address of register, opcode, and immediate value get separated. Data stored in register bank is sent to ALU. Opcode sent to ALU inofrms ALU to perform multiplication operation on data. The product of two 32 bit numbers will be max to max 64 bit in size. So we store the output in two different PIPO registers. In the final stage this product is stored back into the register bank at two consecutive locations (i.e R and R+1). Also at the final stage we simply increment the value of PC so that it points towards the next instruction. Instead of multiplication if we would have a jump instruction then the PC will be loaded with immediate value. Also we had multiplication operation this time so we by passed the RAM module. RAM is only used incase of load / store instruction. The MUX module plays a very important rule. It decides whether data coming from RAM or data coming from ALU has to be stored in register bank.



Working of control path

Now we will be examining the control path in great detail. As mentioned earlier the control path is hardware implementation of moore machine. Same can be observed from the above diagram. The figure given above is called as a state transition diagram. The control path is the brain of the processor. It takes feedback (status flag) from the datapath and generates control signals accordingly. Initially the controller is in the idle state. In this state no control signals are produced. As soon as we power on the processor and give start signal (start = 1), there is change of state from SX to S0. If there is no start signal (start = 0) the controller remains in idle state. In other words we can say that start signal initiates the execution of entire program. Once the processor enters into state S0 it starts behaving as a moore machine. Every clock cycle there is transition taking place from current state to next state. The states go from S0 to S4, and back to S0. This indicates that one instruction has been succesfully executed. This cycle is repeated a number of times untill processor finishes executing all the instructions.

Different control signals and their application -

- Read_i – it decides when to load the next instruction from ROM.
- Read_r – this signal instructs register bank to send its data to ALU.
- Write_r – this signals instruction to enter data into different registers of register bank.
- Read_d – tells RAM to perform load (read_d = 1) or store operation (read_d = 0).
- ALU_on – only when this signal goes high the ALU performs operations on data.
- Signal – only for load instruction signal = 1. For rest all instructions signal = 0.
- Load – incase of jump the PC should be loaded with immediate value
- Increase – incase there is no jump the PC should simply point to next instruction.

Instruction set architecture (ISA) -

Instruction Set Architecture (ISA) is a set of rules and conventions that dictate the behavior and functionality of a computer's CPU. It serves as an interface between hardware and software, allowing software to control the hardware components of a computer system. The ISA defines a set of instructions that a CPU can execute. These instructions include operations like arithmetic (add, subtract, multiply, divide), data movement (load, store), control flow (branch, jump), and more. Each instruction is represented by a unique binary code called as opcode. In our project are having 23 types instructions as of now. Also as mentioned earlier the size of our opcode is 6 bits. This 6 bits are more than sufficient to uniquely identify these 23 instructions. ISA defines the data types and how they are represented and manipulated within the CPU. Common data types include integers, floating-point numbers, and characters. Our processor will be able to handle only positive integer values. To handle floating point operation most of the processor have a co processor called as FPU floating point unit. ISA also talkes about how memory is organized and accessed. This includes addressing modes (e.g., direct, indirect, indexed), memory hierarchy (e.g., cache, RAM, secondary storage), and the byte ordering (e.g., little-endian or big-endian) for multi-byte data. Little-endian stores the least significant byte first, while big-endian stores the most significant byte first. In verilog when we define a vector as [7:0] it means little endian form. Whereas when we define vector as [0:7] it means big endian form. in our design we make use of little endian format.

Classification of instruction based on addressing modes

Add R1 #12	R1 = R1 + 12	Immediate
Add R1 R6	R1 = R1 + RB(6)	Register direct
Add R1 @R6	R1 = R1 + mem(RB(6))	Register indirect
Add R1 54	R1 = R1 + mem(54)	Memory direct
Add R1 @54	R1 = R1 + mem(mem(54))	Memory indirect

In our processor design we have used immediate addressing mode, register direct addressing mode and memory direct addressing modes.

Popular ISAs include x86 (used in many personal computers), ARM (common in mobile devices and embedded systems), MIPS, and PowerPC, among others. Programmers and compilers must adhere to the ISA when writing software for a specific CPU architecture, as it dictates how instructions should be formatted and used. Understanding the ISA is crucial for optimizing code for a particular CPU and for ensuring that software runs efficiently and correctly on a given platform. In this project we have preferred to design our own ISA.

Each instruction in the ISA is encoded as a binary sequence. The encoding format specifies how the bits in an instruction are used to identify the operation, operands, and addressing modes.

Opcode(6)	Source reg(5)	Source reg(5)	Destination reg(5)	Immediate value(11)

Opcode	Format					Operation
000001	ADD	R1	R2	R3		R3 = R1 + R2
000010	SUB	R1	R2	R3		R3 = R1 – R2
000011	AND	R1	R2	R3		R3 = R1 & R2
000100	OR	R1	R2	R3		R3 = R1 | R2
000101	LT	R1	R2			For R1<R3 flag3 = 1, for R1>R3 flag3 = 0
000110	GT	R1	R2			For R1<R3 flag3 = 1, for R1>R3 flag3 = 0
000111	LOAD	K		R3	I.V.	R3 = mem [I.V. + K]
001000	STORE	K	R2		I.V.	mem [I.V. + K] = R2
001001	MOV			R3	57	R3 = 57
001010	RR	R1		R3		R3 = R1 rotated right by 4 bits
001011	RL	R1		R3		R3 = R1 rotated left by 4 bits
001100	JZ	R1			10	PC = 10 (if R1 == 0)
001101	JNZ	R1			10	PC = 10 (if R1 != 0)
001110	JE	R1	R2		10	PC = 10 (if R1 == R2)
001111	JNE	R1	R2		10	PC = 10 (if R1 != R2)
010000	JMP				10	PC = 10
010001	MUL	R1	R2	R3		R3 = R1 * R2
010010	DIV	R1	R2	R3		R3 = R1 % R2
010011	MOD	R1	R2	R3		R3 = R1 / R2
010100	NOT	R1		R3		R3 = ~R1
010101	MOVR	R1		R3		R3 = R1
010110	JFZ				10	PC = 10 (if flag4 = 0)
010111	JFNZ				10	PC = 10 (if flag4 = 1)

Need of Video graphics array (VGA) –

The output of processor can be easily observed on Vivado software. However when we try to run the same code on hardware (FPGA) we face a lot of issues. The data handled by processor has size of 32 bits. This means we require 32 leds to observe output. However FPGA has only 4 onboard leds. Even devices like seven segment display and 16x2 lcd screen would not be able to display such large data. So we decided to use PC monitor to display our data. For interfacing monitor to FPGA we will use VGA protocol.

Working of VGA protocol –

Vga is used to control monitor based on CRT technology. Inside a CRT monitor there is cathode ray beam which is projected at small portion on the phosphorus coated screen. At this point the phosphorus glows which we refer to a pixel. The brightness of the pixel depends on the amount of current. It remains glowing for several 100us even after cathod ray has shifted. Color CRT screens make use of 3 cathod rays i.e red, green, & blue. This cathod ray keeps moving across the entire screen. Theis is done using deflection coils. These coils produce electro magnetic field which bends the cathod ray beam to the desired location on the screen. Theses deflections take place so fast that our eyes cannot make out the difference. We feel that entire screen is getting displayed simultaneously. At top left hand corner of the screen we place the origin (0,0). This is considered as the 1st pixel of the screen. At the bottom right corner we have the last pixel of the screen. When cathod ray completes its journey from 1st pixel to last pixel we say that one frame is completed. The time time taken for this is called refresh rate of the screen. Generally refresh rate is 60Hz. We will be using monitor having 1800 x 795 total pixels. But we never use all the pixels for display. The display area will be 1368 x 768. This is decided using two signals called as horizontal sync and vertical sync. The VGA connector has 15 pins. Out of these only 5 pins are useful to ous. Pins 1,2,3 for RGB respectively. Pins 14, 15 for h-sync, v-sync respectively.

VESA Signal 1368 x 768 @ 60 Hz timing

General timing

Screen refresh rate	60 Hz
Vertical refresh	47.7 kHz
Pixel freq.	85.86 MHz

Horizontal timing (line)

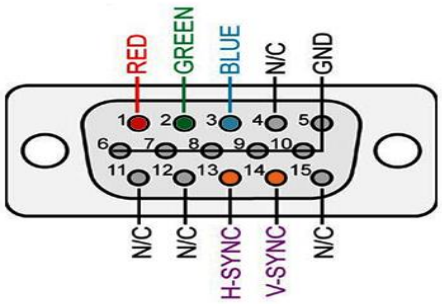
Polarity of horizontal sync pulse is negative.

Scanline part	Pixels	Time [µs]
Visible area	1368	15.932914046122
Front porch	72	0.83857442348008
Sync pulse	144	1.6771488469602
Back porch	216	2.5157232704403
Whole line	1800	20.964360587002

Vertical timing (frame)

Polarity of vertical sync pulse is positive.

Frame part	Lines	Time [ms]
Visible area	768	16.100628930818
Front porch	1	0.020964360587002
Sync pulse	3	0.062893081761006
Back porch	23	0.48218029350105
Whole frame	795	16.666666666667



DETAIL FUNGSI DARI MASING-MASING PIN

1 - Red Video
2 - Green Video
3 - Blue Video
4 - Reserved
5 - GND
6 - Red GND
7 - Green GND
8 - Blue GND
9 - +5 V DC
10 - Sync GND
11 - Monitor ID
12 - DDC SDA
13 - H_{sync}
14 - V_{sync}
15 - DDC SCL

VGA PINOUT

Pixel clock rate = p * l * s = 1800 * 795 * 60 = 85.86 MHz = 85 MHz

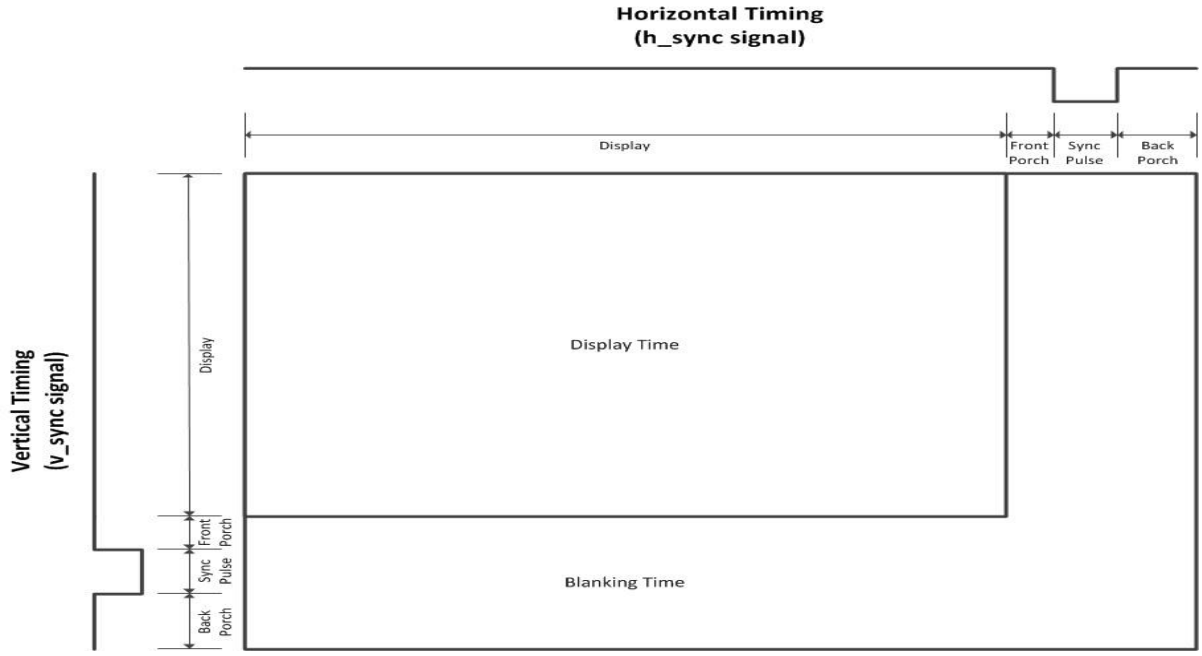
At this speed the vga controller must operate to display data correctly. But the internal clock available on FPGA board is 125 MHz.

Method 1 – dsign clock multiplier & clock divider circuits using verilog.

125 * 2 = 250, 250 % 3 = 83.33, this value is close to 85 MHz

We succesfully designed clock multiplier but failed to design clock divider circuit.

Method 2 – there are ready made circuits available on vivado software which can be instantiated into our design. Such circuits are called as IP. One such IP is clock wizard. It helps you to generate internal clock of desired frequency without writing a single line of vrilog code. We used it to generate 85 MHz clock form 125MHz available clock.



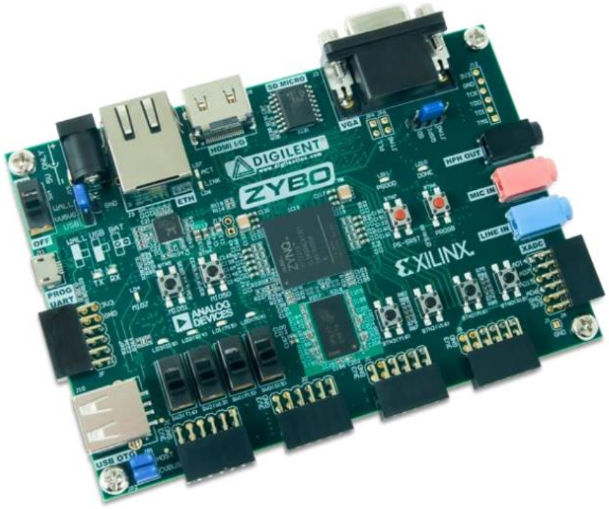
Display ASCII characters on VGA –

ASCII stands for american standard code for information interchange. Each alphabet, number, special symbol is assigned a unique ASCII code. For eg ASCII code of letter A is 65 in decimal (41 in hexadecimal). To display one ASCII character we require 16 row and 8 column i.e 128 pixels (16*8=128). We have total 1368 pixel in a row. This means we can display 171 ASCII characters in a row. Similarly we have 768 pixels in column. This means we can display 48 ASCII characters in a column. In total we can display 8208 ASCII characters on the screen at a single time. Each ASCII character has 11 bit address. First 7 bits decide which character to print. Last 4 bits decide which line to print among 16 lines of a character.

CHAPTER 4
HARDWARE & SOFTWARE

HARDWARE –

ZYBO Zynq – 7000 board by Digilant



The ZYBO Zynq-7000 development board.

PC monitor

VGA connector cable

Keyboard

PS2 connector cabel

SOFTWARE –

Xilinx Vivado

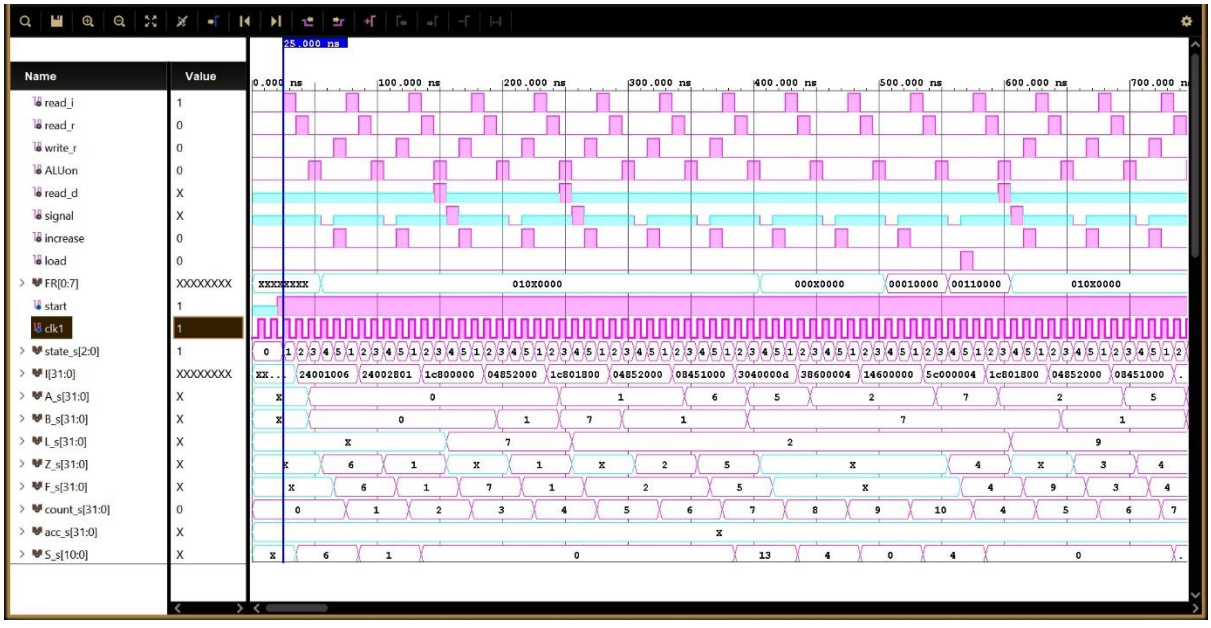
CHAPTER 5
RESULT & DISCUSSION

RESULT

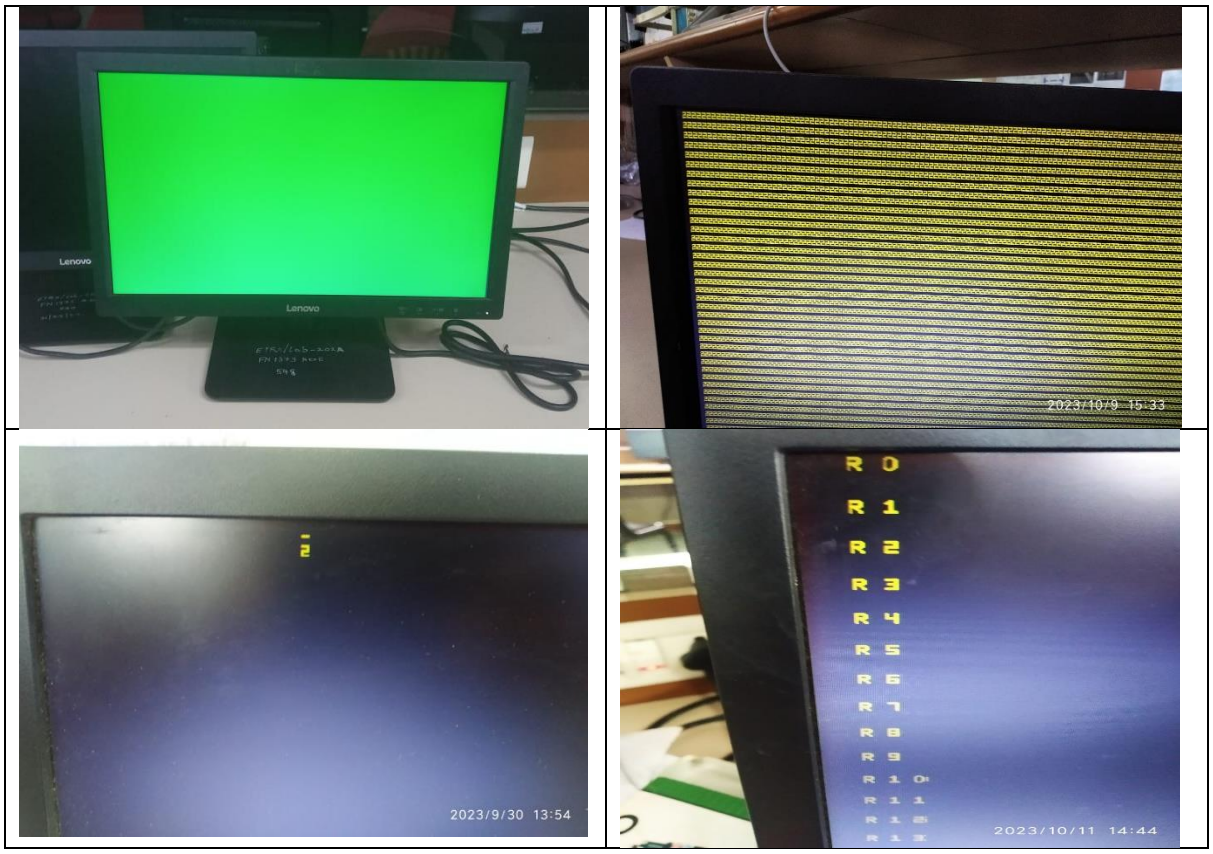
We completed the entire design of RISC processor on VIVADO software. We tested the processor by writing a testbench for it. A simple assembly level program was written to find largest number stored in RAM. 6 different values were stored in RAM. During simulation it was observed that our processor was able to do the required computation and give accurate results.

```
initial begin
        #0 clk1 = 0;


        #0 DP.RAM.ram_mem[0] = 7;
           DP.RAM.ram_mem[1] = 2;
           DP.RAM.ram_mem[2] = 9;
           DP.RAM.ram_mem[3] = 14;
           DP.RAM.ram_mem[4] = 45;
           DP.RAM.ram_mem[5] = 23;

        #0 for(k=0; k<32; k=k+1) begin DP.RB.regbank[k]=0; end

           DP.ROM.rom_mem[0] = 32'h24001003;  // MOV - - R2 3
           DP.ROM.rom_mem[0] = 32'h24001006;  // MOV - - R2 6
           DP.ROM.rom_mem[1] = 32'h24002801;  // MOV - - R5 1
           DP.ROM.rom_mem[2] = 32'h1C800000;  // LOAD R4 - R0 0
           DP.ROM.rom_mem[3] = 32'h04852000;  // ADD R4 R5 R4 -
           DP.ROM.rom_mem[4] = 32'h1C801800;  // LOAD R4 - R3 0
           DP.ROM.rom_mem[5] = 32'h04852000;  // ADD R4 R5 R4 -
           DP.ROM.rom_mem[6] = 32'h08451000;  // SUB R2 R5 R2 -
           DP.ROM.rom_mem[7] = 32'h3040000D;  // JZ R2 - - 13
           DP.ROM.rom_mem[8] = 32'h38600004;  // JE R3 R0 - 4
           DP.ROM.rom_mem[9] = 32'h14600000;  // LT R3 R0 - -
           DP.ROM.rom_mem[10] = 32'h5c000004; // JFNZ - - - 4
           DP.ROM.rom_mem[11] = 32'h54600000; // MOVR R3 - R0 -
           DP.ROM.rom_mem[12] = 32'h40000004; // JMP - - - 4
           DP.ROM.rom_mem[13] = 32'h07C00800; // ADD R30 R0 R1 -
```

```
Tcl Console
# run 1000ns
                 20ns>       0, R1=       0, R2=       0, R3=       0, R4=       0, R5=       0
                 75ns>       0, R1=       0, R2=       6, R3=       0, R4=       0, R5=       0
                125ns>       0, R1=       0, R2=       6, R3=       0, R4=       0, R5=       1
                175ns>       7, R1=       0, R2=       6, R3=       0, R4=       0, R5=       1
                225ns>       7, R1=       0, R2=       6, R3=       0, R4=       1, R5=       1
                275ns>       7, R1=       0, R2=       6, R3=       2, R4=       1, R5=       1
                325ns>       7, R1=       0, R2=       6, R3=       2, R4=       2, R5=       1
                375ns>       7, R1=       0, R2=       5, R3=       2, R4=       2, R5=       1
                625ns>       7, R1=       0, R2=       5, R3=       9, R4=       2, R5=       1
                675ns>       7, R1=       0, R2=       5, R3=       9, R4=       3, R5=       1
                725ns>       7, R1=       0, R2=       4, R3=       9, R4=       3, R5=       1
                975ns>       9, R1=       0, R2=       4, R3=       9, R4=       3, R5=       1
INFO: [USF-XSim-96] XSim completed. Design snapshot 'testbench_4_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:05 ; elapsed = 00:00:11 . Memory (MB): peak = 2539.875 ; gain = 0.000
run all
                1075ns>       9, R1=       0, R2=       4, R3=      14, R4=       3, R5=       1
                1125ns>       9, R1=       0, R2=       4, R3=      14, R4=       4, R5=       1
                1175ns>       9, R1=       0, R2=       3, R3=      14, R4=       4, R5=       1
                1425ns>      14, R1=       0, R2=       3, R3=      14, R4=       4, R5=       1
                1525ns>      14, R1=       0, R2=       3, R3=      45, R4=       4, R5=       1
                1575ns>      14, R1=       0, R2=       3, R3=      45, R4=       5, R5=       1
                1625ns>      14, R1=       0, R2=       2, R3=      45, R4=       5, R5=       1
                1675ns>      45, R1=       0, R2=       2, R3=      45, R4=       5, R5=       1
                1975ns>      45, R1=       0, R2=       2, R3=      23, R4=       5, R5=       1
                2025ns>      45, R1=       0, R2=       2, R3=      23, R4=       6, R5=       1
                2075ns>      45, R1=       0, R2=       1, R3=      23, R4=       6, R5=       1
                2325ns>      45, R1=       0, R2=       1, R3=       0, R4=       6, R5=       1
                2375ns>      45, R1=       0, R2=       1, R3=       0, R4=       7, R5=       1
                2425ns>      45, R1=       0, R2=       0, R3=       0, R4=       7, R5=       1
                2525ns>      45, R1=      45, R2=       0, R3=       0, R4=       7, R5=       1
$finish called at time : 10000 ns : File "C:/Users/Ramesh/Desktop/FPGA/sample_2/sample_2.srcs/sim_1/new/testbench_4.v" Line 97
```



Now we know that our processor works fine on software, it is time to test this processor on actual hardware. Just because code runs on software it does not mean it will also run on hardware. We have designed entire processor in behavioural level and not structural level. This might create some problem while running the code on hardware. Hence it is necessary to test the code on FPGA. To do this we will first have to interface input and output devices to FPGA, i.e keyboard and monitor. Keyboard to write assembly level codes and monitor to view the output. We have completed the task of interfacing FPGA board to monitor. Here are the results.



DISCUSSION

Feedback given by review committee during review one (26th august 2023)

- To change the title of project. Title is very short and incomplete.
- Add only those softwares in presentation which you know how to operate. You won't be able to answer if external puts a question on these softwares.
- What should be the initial value stored in every register of register bank. When ever we turn on processor or reset the processor all the memory locations should be initialised to zero and not have some garbage value.
- What is the difference between your RISC processor and the existing RISC processor. You are trying to improve which parameter of the processor power consumption, processing speed, heat dissipiation, chip area, design complexity.
- Why are you performing sign extension on immediate value (11 bit to 32 bit). 11 bit address BUS is used for 1Kb memory (2^10 = 1Kb). For 1Kb memory 10 bits are sufficient. ROM and RAM are of 1Kb each which is wrong.
- How is instruction being encoded and decoded in the processor. Add a separate module for instruction decoder.
- Your processor performs operations on which all data types. Integers, floating point numbers, fixed point numbers etc.
- Add more number of instructions like multiplication, division, not gate etc.
- A flag for LT & GT instructions was missing. Sir told to add separate flag for less then and greater than instructions.

Feedback given by review committee during review two (14th october 2023)

- In last review sir told to add encoder and decoders for instruction. So we added a 1:5 decoder. However sir told that it is not standard encoder format. Go for designing 1:8 decoder and keep three of the outputs undefined. They can be later used if necessary.
- Sir asked ous howwill us handle the product of multiplication. During multiplication operation the biggest challenge is storing the output. We told sir that we have created two PIPO register named 'z' and 'accumulater' which store the lower and upper nibble of 64 bit output. These registers then store the value in the registerbank at location R and R+1. (two consecutive locations inside register bank).
- The product of multiplication is always store in 2 consecutive locations. (R and R+1). But there is a limit to the number of registers present in register bank. Our RB contains only 32 registers. Suppose R=32 then R+1=33 register which doesnot exist.
- We had used 2 separate instruction to calculate quotient and remainder of divison. Sir suggested us to keep only one instruction and not 2 separate instruction. RISC stands for reduced instruction set architecture. So the aim must be to have least possible instruction.
- Initially we had 4 sepearate flags. But we converted it into 8 bit flag register. Out of this 4 flags are defined and 4 are undefined. But sir told ous to go for 32 bit flag register. This will make it easy for ous to put all registers like PC, IR, FR, etc inside the RAM.
- Our aim is to show data present inside 32 registers on PC monitor. Instead sir suggested ous to show the contents of RAM on the monitor. Just like in the case of 8051 microcontroller where all memory location are visible.

CHAPTER 6
CONCULUSION & FUTURE SCOPE

CONCULUSION

This is an year long project. The progress made so far has been explained in this project report. By the end of this academic year we aim to complete this project. Out of three major tasks we have completed two. Processor design & monitoir interface is done. Keyboard interface is yet to be done.

During this period we learned a lot about different types of architectures. Like the RISC, CISC & ARM architecture. How different digital circuits can be designed using verilog language. We understood that there are two types of design that is semi custom design and full custom design. The design you choose depends on what is your target hardware FPGA or ASIC. We aslo studied the internal architecture of FPGA boards in general. Like the gates, flip flops, DSP, LUT, BRAM etc. We gained experties in working on VIVADO softwares which is used in industries. Understood the flow of design in VIVADO software –

- Algorithm/ truth table/ state transition diagram
- Verilog code + constraint file
- Synthesis
- Implementation
- Generate bit stream

While interfacing FPGA to monitor we studied about the VGA protocol. How horizontal sync & vertical sync signals control the display area of monitor. How group of pixels can be used to display ASCII characters. The RGB signals each of one bit can produce a maximum of 8 different colors (2^3 = 8). To increase the depth of color we can increase the number of bit per signal. For example if RGB are 3 bits each then we can display total 512 colors.

FUTURE SCOPE

If this project completes on time, we will try to modify this design. We will introduce pipeling concept in our processor. Since our processor require 5 T states to complete one instruction we will introduce 5 stage pipelining. Pipeling will drastically improve the speed and efficiency of the processor. At the same time we will require a lot more hardware to handle the exception. We will have to take care of the 3 pipelining hazards. Those are structural hazard, data hazard and control hazard.

Apart from this we also aim to go for ASIC design. Using softwares like CADENCE we can design the processor and make a chip out of it. CADENCE software is also one of the widely used softwares in industry. Learning CADENCE will give ous more hands on experience on ASIC design.

Our RISC processor is a general purpose computing device. It performs ordinary arithmatic and logical operation. We aim to upgrade it and make it application specific RISC processor. It should be able to solve real world problems and have some application in the field of medicine, IOT, AI, ROBOTICS etc.

CHAPTER 7
REFERENCES

REFERENCES

Research papers -

- (T. Zheng, G. Cai and Z. Huang, "A Soft RISC-V Processor IP with High-performance and Low-resource consumption for FPGA," 2022 IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, 2022, pp. 2538-2541, doi: 10.1109/ISCAS48785.2022.9937742.)
- (Urquhart, Roddy (29 March 2021). "What Does RISC-V Stand For? A brief history of the open ISA". Systems & Design: Opinion. Semiconductor Engineering.)
- ("The History of the Integrated Circuit". Nobelprize.org. Archived from the original on 29 Jun 2018. Retrieved 21 Apr 2012.)
- (Pawson, Richard (30 September 2022). "The Myth of the Harvard Architecture". IEEE Annals of the History of Computing. 44 (3): 59–69. doi:10.1109/MAHC.2022.3175612. S2CID 252018052)
- "Design Of 32 Bit Asynchronous RISC-V Processor Using Verilog", International Journal of Emerging Technologies and Innovative Research (www.jetir.org | UGC and issn Approved), ISSN:2349-5162, Vol.7, Issue 3, page no. pp1717-1722, March-2020,
- International Research Journal of Engineering and Technology (IRJET) Volume: 08 Issue: 06 | June 2021 . Shruthi1, Dr. Jamuna S2 1PG Student (M.Tech in VLSI Design and Embedded Systems), Department. of ECE, DSCE Bangalore, Karnataka 2Professor, Department. of ECE, DSCE, Bangalore, Karnataka
- Mr. Rajkumar D. Komati1, Aditya Kolekar2, Kunal Kasbekar3, Ms. Avanti Godbole4 1Assistant Professor, Dept. of ENTC, MIT College of Engineering, Kothrud, Pune 2, 3, 4Undergraduate Student Dept. of ENTC, MIT College of Engineering, Kothrud, Pune.
- . International Journal for Research in Applied Science & Engineering Technology (IJRASET) ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538 Volume 11 Issue IV Apr 2023- Available at www.ijraset.com Authors: Manjula B B, Vidyashree H, Kavyashree M G, Tulasi V, Arpitha R

Websites –

- https://chat.openai.com/
- https://www.chipverify.com/tutorials/verilog
- https://www.fpgadeveloper.com/list-of-fpga-dev-boards-by-vendor/
- https://medium.com/programmatic/how-to-design-a-risc-v-processor-12388e1163c
- https://en.wikipedia.org/wiki/Bitwise_operation#Circular_shift
- https://zipcpu.com/dsp/2017/12/14/logic-pll.html
- https://esrd2014.blogspot.com/p/register-file.html
- https://courses.cs.washington.edu/courses/cse370/10sp/pdfs/lectures/regfile.txt
- https://hackaday.io/project/20864-ygrec-15-bis/log/56857-the-program-counters-circuit

youtube videos –

- https://www.youtube.com/watch?v=4enWoVHCykI&list=WL&index=17&pp=gAQBiAQB
- https://www.youtube.com/watch?v=KrhllV003Vs&list=WL&index=19&pp=gAQBiAQB
- https://www.youtube.com/watch?v=v1CoZZiJKzc&list=WL&index=20&pp=gAQBiAQB
- https://www.youtube.com/watch?v=DqyOiPUn_LM&list=WL&index=27&pp=gAQBiAQB
- https://www.youtube.com/watch?v=w8ZTlsle6q8&list=WL&index=28&pp=gAQBiAQB
- https://www.youtube.com/watch?v=BcJU234Jh3k&list=WL&index=31&pp=gAQBiAQB
- https://www.youtube.com/watch?v=e6j7AR-htB0&list=WL&index=37&pp=gAQBiAQB
- https://www.youtube.com/watch?v=ngkpvMaNapA&list=WL&index=44&pp=gAQBiAQB
- https://www.youtube.com/watch?v=JFuzqRTOwxo&list=WL&index=83&pp=gAQBiAQB
- https://www.youtube.com/watch?v=Ngu1UbRAeqQ&list=WL&index=100&pp=gAQBiAQB
- https://www.youtube.com/watch?v=NCrlyaXMAn8&list=PLJ5C_6qdAvBELELTSPgzYkQg3HgclQh-5&pp=iAQB
- https://www.youtube.com/watch?v=MVu5OAFZhKA&list=WL&index=26&pp=gAQBiAQB