Design and Implementation of Five Stages Piplined RISC Processor on FPGA

1st Amani Najeeb Ben Yousuf
Department of Computer Engineering
University of Tripoli
Tripoli, Libya
a.yousuf@uot.edu.ly

2nd Dania Khaled Alamen
Department of Computer Engineering
University of Tripoli
Tripoli, Libya
d.alamen@uot.edu.ly

3rd Mohamed Muftah Eljhani
Department of Computer Engineering
University of Tripoli
Tripoli, Libya
m.eljhani@uot.edu.ly

***Abstract*— This research focuses on designing and implementing a processor with a five-stage pipeline for educational purposes. The proposed processor can execute five 16-bit instructions simultaneously and is designed and simulated using Verilog HDL. It is implemented on a Cyclone IV FPGA on the DE2i-150 design board. The processor is suitable for applications that require high processing speed and low power consumption, including mobile computers, consumer electronics, security systems, and more. This RISC-based processor has simple instructions that consume low power and execute quickly.**

Keywords— The areas of expertise include designing processors using Pipelined RISC architecture, designing systems on programmable chips, and creating FPGA designs using the Verilog hardware description language.

I. INTRUDUCTION

The evolution of computer technology has witnessed the emergence of various architectures since the advent of first-generation computers in the 1940s. Over time, efforts have been made to enhance computer performance. One popular technique used in current CPU designs, microprocessors, and microcontrollers is instruction pipelining, which can significantly increase the number of instructions executed in a specific time interval [1][2]. Our design philosophy for the architecture was based on the RISC principle of favoring a smaller and simpler set of instructions that could be executed in the same amount of time. This led us to maintain a highly simplified instruction set [3]. Pipelining is a fundamental aspect of RISC processors that mimics the workflow of a manufacturing assembly line. By processing different stages of the instruction simultaneously, the processor can execute more instructions in a shorter span of time [4]. RISC processors are preferred over CISC processors when it comes to implementing pipelining. This is because traditional instruction cycles of CISC processors tend to waste CPU resources by including other services, such as reading or writing to memory or input/output devices, which leave the CPU idle. To improve instruction latency and program throughput, pipelining has proven to be highly efficient. As computer systems continue to evolve, technological advances such as speed up circuits and improved organization, including the addition of instruction pipelines to processors, are utilized to achieve higher performance [5][6][7]. The pipeline processor operates by initially placing the first instruction in the decode stage. Subsequently, the second instruction is fetched while the first instruction is in the execute stage. Once the first instruction completes execution after three cycles, the second instruction moves to the decode stage and the third instruction is fetched. Thereafter, instructions are completed every cycle

Non-pipelined systems involve an instruction cycle that requires three separate stages before moving on to the next instruction, and it is not always possible to begin executing the next set of instructions on each clock cycle. This can lead to hazards, which are complications that can arise during instruction execution. Common types of hazards include structural, data, and control hazards [8]. In 1997, an educational software called WinDLX was developed as a pipeline simulator for the DLX processor. It was created using C++ and designed for MS Windows (16-bit) platforms. The simulator's model is primarily based on the DLX architecture proposed by Hennessy Patterson, focusing on the architectural aspect. WinDLX was developed to assist in teaching the concept of instruction pipeline to students [9].

Another approach to pipeline simulation is using the ModelSim simulator and the Xilinx ISE tools software, as described in [2]. This method involves dividing the pipeline processing into distinct phases (fetch, decode, execute, and store) using Verilog HDL, and designing each of them accordingly. A detailed step-by-step scheme is provided, which helps users better grasp the underlying concepts of the processor. However, utilizing these tools requires some expertise in simulation techniques. Another related work involves using Java programming for pipeline simulation, with a focus on student engagement and interactivity [10]. The RISC-16 processor was selected for its simplicity, comprehensiveness, and suitability for educational use. The team's system allowed users to create their own assembly language programs and visually visualize the processor's internal dynamic behavior. Additionally, the team's design has the potential to serve as a web-based simulation model that can explore the state space defined by the integrated parallel model and simulator [8][11]. The researchers demonstrated that the simulator has the capability to function as a calculator and accurately calculate acceleration based on given input parameters. Furthermore, by utilizing the unified parallel model and simulator, users are able to assess the potential for parallel speedup continuity across various computer architectures with hardware support at multiple levels of parallelism. The simulation tools enable the customization of architecture and network configurations and the ability to run diverse workloads with variable configurations. Our objective is to develop a pipelined RISC processor that contains five stages and supporting eight instructions, with a focus on achieving high performance. Subsequently, we aim to implement our design on DE2i-150 FPGA board.

II. METHODOLOGY

The primary objective of this investigation is to present a pipeline design with five stages to execute a RISC 16-bit instruction set processor. The system can be divided into different stages, each of which has a specific function, such as Instruction Fetch (IF), Instruction Decode (ID), Instruction Execute (EX), Memory Access (MEM), and Write Back (WB).

A. Top Module View

The Pipelined CPU architecture proposed in "Fig. 1" comprises of six subordinate modules.

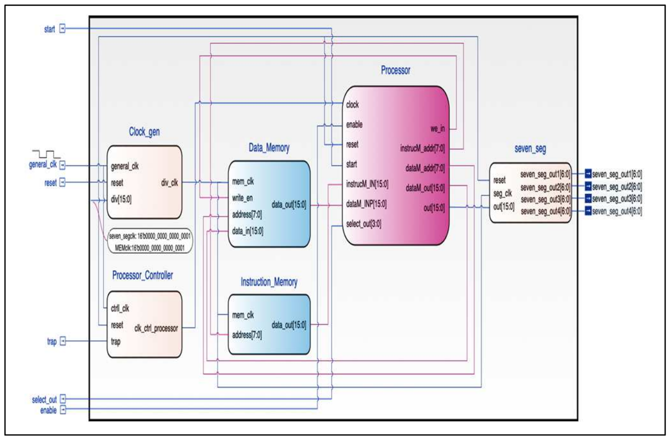


Fig. 1. System Top View

B. System Sub-Modules

- Processor Main Module: Arithmetic processing is carried out using the instruction retrieved from the instruction memory, and data is either written to or read from the data memory.
- Instruction Memory Module: Provide instructions for the processor module, so it can find the memory address of the instruction through the value in the instruction register, thereby reading the instruction.
- Data Memory Module: Processor reads data of writes data.
- Clock Generator Module: An electronic oscillator that is used to synchronize by producing a clock signal.
- Controller Module: It is used to control the processor execution by the clock.
- Seven-Segment Interface Module: An electronic display device that presents decimal numerals and serves as a simpler option to dot matrix displays is called a numeric display.

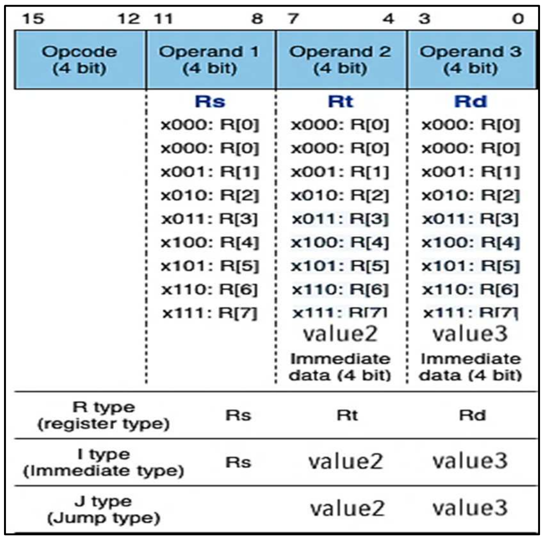
C. Instruction Set Architecture

The pipelined RISC processor's Instructions are divided into four categories:

1. Arithmetic and logical instructions that carry out arithmetic and logical operations.
2. Data movement instruction (Load and Store Instructions) are used to move data between the accumulator, general registers, and memory. These instructions facilitate the transfer of data within the system.
3. Jump instruction is used to alter the sequence in which instructions are executed
4. Miscellaneous instruction.

"Table. I". show the pipelined instruction format, and "Table. II". show the instruction encoding.

TABLE. I. Pipelined Instruction Format



15 12	11 8	7 4	3 0
Opcode (4 bit)	Operand 1 (4 bit)	Operand 2 (4 bit)	Operand 3 (4 bit)
	Rs	Rt	Rd
	x000: R[0]	x000: R[0]	x000: R[0]
	x000: R[0]	x000: R[0]	x000: R[0]
	x001: R[1]	x001: R[1]	x001: R[1]
	x010: R[2]	x010: R[2]	x010: R[2]
	x011: R[3]	x011: R[3]	x011: R[3]
	x100: R[4]	x100: R[4]	x100: R[4]
	x101: R[5]	x101: R[5]	x101: R[5]
	x110: R[6]	x110: R[6]	x110: R[6]
	x111: R[7]	x111: R[7]	x111: R[7]
		value2 Immediate data (4 bit)	value3 Immediate data (4 bit)
R type (register type)	Rs	Rt	Rd
I type (Immediate type)	Rs	value2	value3
J type (Jump type)		value2	value3

TABLE. II. 16-bit Instruction Set Encoding

Instr.	Oper and1	Oper and2	Opera nd3	Opcode	Operation
Arithmetic and logical instructions					
ADD	Rs	Rt	Rd	#1000	$Rs \leftarrow Rt + Rd$
SUB	Rs	Rt	Rd	#1010	$Rs \leftarrow Rt - Rd$
AND	Rs	Rt	Rd	#1101	$Rs \leftarrow Rt\ and\ Rd$
XOR	Rs	Rt	Rd	#1111	$Rs \leftarrow Rt\ xor\ Rd$
Data movement instructions					
LOAD	Rs	Rt	value3	#0010	$R[Rs] \leftarrow m[Rt + value3]$
STORE	Rs	Rt	value3	#0011	$m[Rt + value3] \leftarrow Rs$
Jump instruction					
JUMP		value 2	value3	#0110	$jump\ to\ \{value2, value3\}$
Miscellaneous					
HALT				#0001	$halt$

D. *Five Stages Pipelined Procedure*

Besides the instruction set, the most important part in CPU design is the implementation of the system architecture as shown in the block diagram of the RISC. The pipelined architecture implementation of the processor relied on "Fig. 2". "Fig. 3" shows the instruction execution procedure, starting from fetching the instruction, decoding, execution, read/write memory, write back, and hazard detection.

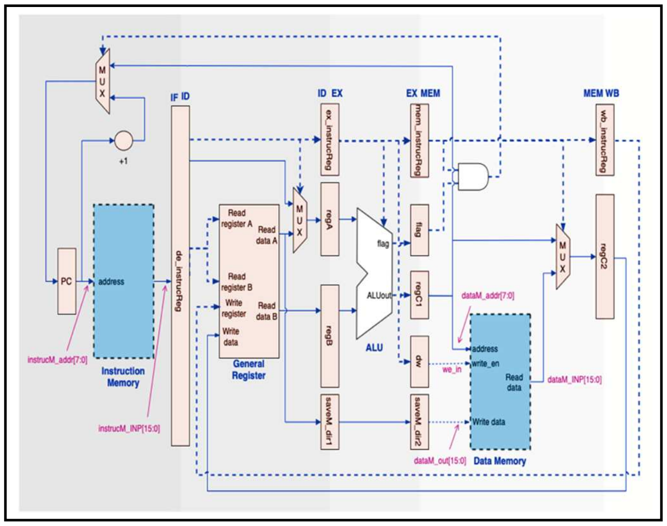


Fig. 2. Pipelined RISC Architecture

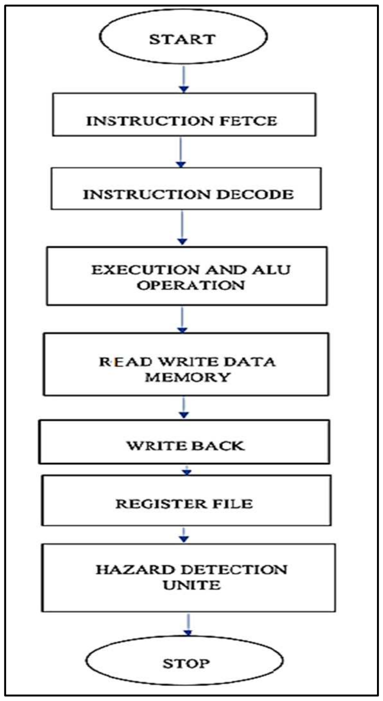


Fig. 3. Pipelined Instruction Procedure

E. *RISC Control Unit*

The CPU control unit is naturally based on the finite state machine. There are only two main states: idle and execute. In the idle state, the CPU can enter the execute state only if enable and start is enabled at the same time, otherwise the CPU will stay in the idle state.

The execute state will go to the idle state as shown in "Fig. 4", if enable is zero or it is in the halt instruction, otherwise it will stay in the execution state. The processor controller is used to control the running time of our processor. It consists of a clock, reset and trap as input. The trap is work as the clock of the processor controller, and the output of the model clk-ctrl-processor gives a specific synchronization to the system.

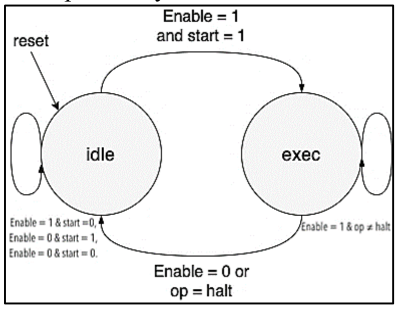


Fig. 4. Control State Diagram

F. *Instruction Fetch*

The IF stage fetches the program instructions one by one. pointing the memory by program counter (PC) value. The program counter keeps track of fetched instructions. Instructions are fetched from memory every clock cycle.

G. *Instruction Decode*

This unit reads commands from the command register and encode the operands by the operation. A 16-bit instruction is divided into several stages as mentioned above.

H. *Instruction Execute*

During the EX-stage, the CPU carries out arithmetic logic unit (ALU) operations as instructed by the decoder and based on the values stored in the flag register.

I. *Arithmetic Logic Unit*

An ALU, or Arithmetic Logic Unit, is a digital circuit that combines different bits of binary numbers to perform arithmetic and bitwise operations on integers in computing.

TABLE. III. Arithmetic Logic Unit Interface

Signal name [size in bits]	**Description**
regB [16]	Input data "regB", from Reg [Rs] in most cases
regA [16]	Input data "regA", from Reg [Rt] or the immediate in most cases.
ALUout [16]	Output data
Opcode [4]	Generated by the control unit when the instruction currently in the ALU was in Decode

Identify applicable funding agency here. If none, delete this text box.

J. Memory Access

When reading or writing data from a data memory, this stage will be used. This stage is only used for loading Memory instructions to read and write data memory. ALU results can be stored directly in data memory. This unit interfaces with the data memory. Memory is the storage used to store runtime instructions (program) and data. In order to distinguish it from the long-term storage of data and programs in a computer, memory that sometimes-called main memory. The Instruction memory stores all the prefetched instructions. It's a combination logic, according to the address read, outputs of an instruction. It does need one read port, which will have 16-bit width (fetch one instruction at a time). The instruction memory can model to having arbitrarily fast asynchronous reads.

TABLE. VI. Instruction Memory Interface

Signal Name [size in bits]	Description
Address [16]	Current instruction to fetch
Read Data [16]	Current instruction fetched

Three ports are required to write data to memory, read data from memory, and address the memory. Data input/output requires 16 bits to match the size of the register. There are two other ports, one for enabling or disabling writes and the other for the clock signal.

A. Reading data is a combinational logic, which directly reads the output of an instruction according to the address.
B. Writing to Data is sequential logic, and may be written once per cycle. To write according to the we-in signal, then the clock frequency of the Memory needs to be faster than the CPU clock.

TABLE. V. Data Memory Interface

Signal name [size in bits]	Description
dataM_INP [16]	Data in Mem[address]. Output is still in the M stage if read asynchronously.
dataM_addr [8]	Address to perform the next read/write.
dataM_out [16]	Data to write to Mem[address].
we_in	Set high if the unit should perform a write on the next clock edge.
Div_clk	Clock signal to synchronize writes.

K. Write Back

Write back writes the calculation result to the value of the instruction. Except for the jump instruction and the load and store instructions, the first operand (register) is written back. and just keep the register unchanged in other cases.

L. Hazard Conflicts

In pipeline processing, due to the dependencies of various stages and the competition of hardware resources, the operation cannot be performed at the same time. The cause of pipeline failure is called a hazard. Hazard is divided into three types [11]:

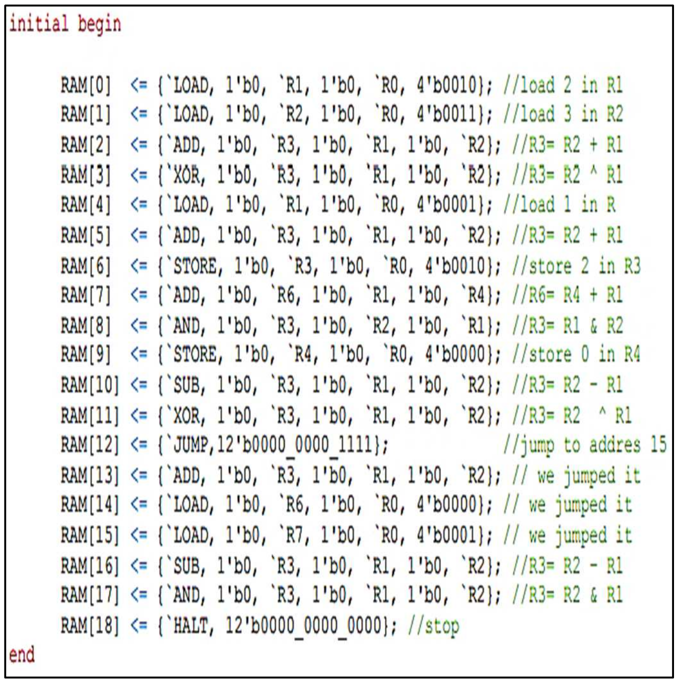
- Structural hazard
- Data hazard
- Control hazard

M. Clock Generator

To enable us to observe the CPU's intermediate actions, we employed a clock divider to reduce the built-in 50MHz clock frequency used for our system clock.

III. SIMULATION AND RESULTS

The system was designed and simulated individually, then instantiated, and verified as top-level module. After the system functionality was verified, "Fig. 5", shows the assembly program and used to verify the system, "Fig. 6", show the input/output simulation result. The resulting simulation waveforms shows that the system is performing all instructions used to test the system. In "Fig. 8", Intel Altera Quartus II software tool used to place and route design into FPGA board, and generate the FPGA fitter report that shown in "Fig. 7".



```
initial begin

     RAM[0]  <= {`LOAD, 1'b0, `R1, 1'b0, `R0, 4'b0010}; //load 2 in R1
     RAM[1]  <= {`LOAD, 1'b0, `R2, 1'b0, `R0, 4'b0011}; //load 3 in R2
     RAM[2]  <= {`ADD, 1'b0, `R3, 1'b0, `R1, 1'b0, `R2}; //R3= R2 + R1
     RAM[3]  <= {`XOR, 1'b0, `R3, 1'b0, `R1, 1'b0, `R2}; //R3= R2 ^ R1
     RAM[4]  <= {`LOAD, 1'b0, `R1, 1'b0, `R0, 4'b0001}; //load 1 in R
     RAM[5]  <= {`ADD, 1'b0, `R3, 1'b0, `R1, 1'b0, `R2}; //R3= R2 + R1
     RAM[6]  <= {`STORE, 1'b0, `R3, 1'b0, `R0, 4'b0010}; //store 2 in R3
     RAM[7]  <= {`ADD, 1'b0, `R6, 1'b0, `R1, 1'b0, `R4}; //R6= R4 + R1
     RAM[8]  <= {`AND, 1'b0, `R3, 1'b0, `R2, 1'b0, `R1}; //R3= R1 & R2
     RAM[9]  <= {`STORE, 1'b0, `R4, 1'b0, `R0, 4'b0000}; //store 0 in R4
     RAM[10] <= {`SUB, 1'b0, `R3, 1'b0, `R1, 1'b0, `R2}; //R3= R2 - R1
     RAM[11] <= {`XOR, 1'b0, `R3, 1'b0, `R1, 1'b0, `R2}; //R3= R2  ^ R1
     RAM[12] <= {`JUMP,12'b0000_0000_1111};              //jump to addres 15
     RAM[13] <= {`ADD, 1'b0, `R3, 1'b0, `R1, 1'b0, `R2}; // we jumped it
     RAM[14] <= {`LOAD, 1'b0, `R6, 1'b0, `R0, 4'b0000}; // we jumped it
     RAM[15] <= {`LOAD, 1'b0, `R7, 1'b0, `R0, 4'b0001}; // we jumped it
     RAM[16] <= {`SUB, 1'b0, `R3, 1'b0, `R1, 1'b0, `R2}; //R3= R2 - R1
     RAM[17] <= {`AND, 1'b0, `R3, 1'b0, `R1, 1'b0, `R2}; //R3= R2 & R1
     RAM[18] <= {`HALT, 12'b0000_0000_0000}; //stop
end
```

Fig. 5. Assembly Code

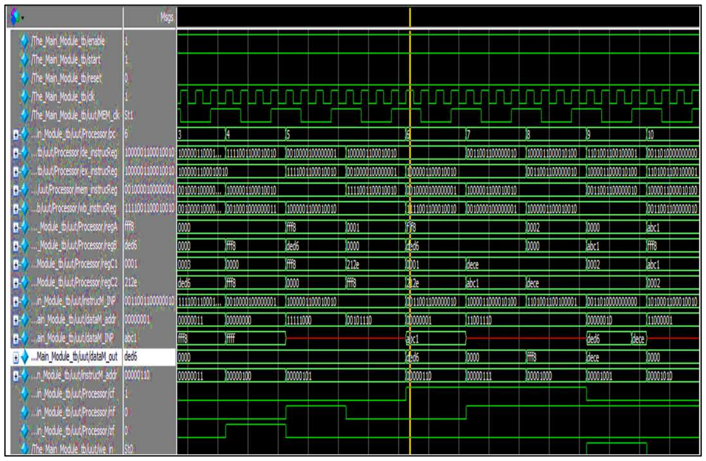
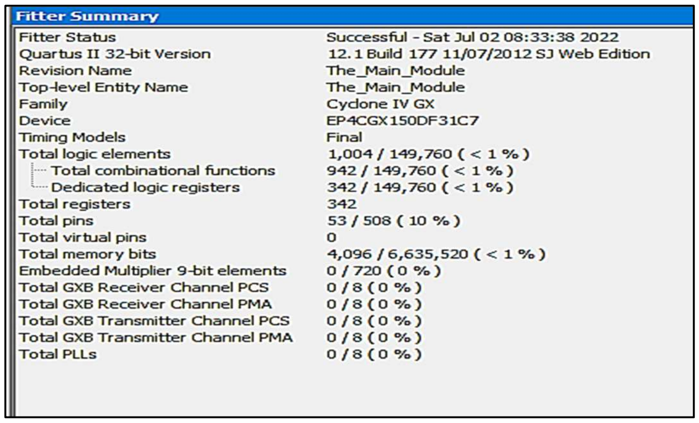


Fig. 6. Simulation Waveforms



Fitter Summary

Fitter Status	Successful - Sat Jul 02 08:33:38 2022
Quartus II 32-bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	The_Main_Module
Top-level Entity Name	The_Main_Module
Family	Cyclone IV GX
Device	EP4CGX150DF31C7
Timing Models	Final
Total logic elements	1,004 / 149,760 (< 1 %)
Total combinational functions	942 / 149,760 (< 1 %)
Dedicated logic registers	342 / 149,760 (< 1 %)
Total registers	342
Total pins	53 / 508 (10 %)
Total virtual pins	0
Total memory bits	4,096 / 6,635,520 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 720 (0 %)
Total GXB Receiver Channel PCS	0 / 8 (0 %)
Total GXB Receiver Channel PMA	0 / 8 (0 %)
Total GXB Transmitter Channel PCS	0 / 8 (0 %)
Total GXB Transmitter Channel PMA	0 / 8 (0 %)
Total PLLs	0 / 8 (0 %)

Fig. 7. FPGA Fitter Summary

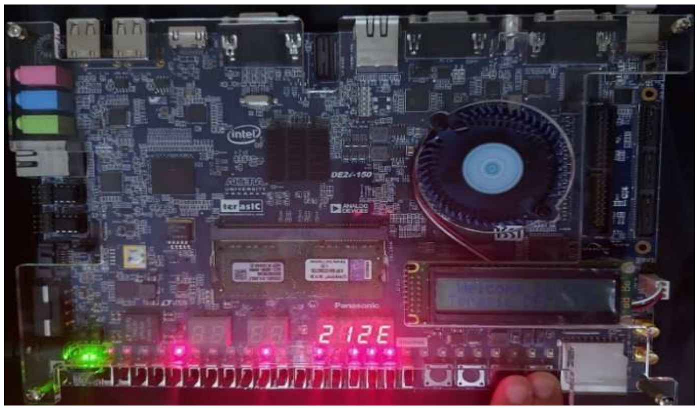


Fig. 8 Pipelined RISC on FPGA Board Fitter

IV. CONCLUSION

The aim of this endeavor is to create an educational 5-stage pipelined processor, which involves its design, validation, and implementation. To determine the processor's efficiency and speed, a series of customized arithmetic, logic, data movement, and branch instructions were employed for analysis. Verilog HDL was utilized to design and simulate the system, which was then mapped to the Cyclone IV FPGA. While executing the pipelined processor, data hazard issues emerged, which were resolved by integrating load utilization modules and data transfer.

REFRENCES

[1] Rakesh MR, Ajeya B, Mohan AR. Novel architecture of 17-bit address RISC CPU with pipelining technique using Xilinx in VLSI Technology. International Journal of Engineering Research and Applications.2014; 4(5):116-121.

[2] Rakesh MR. Design and simulation of four stage pipelining architecture using the Verilog. International Journal of Science and Research. 2014; 3(3):108-12.

[3] Mohamed M. Eljhani, Veton Z. Kepuska, Reduced Instruction Set Computer Design on FPGA. 2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA, 25-27 May 2021, Tripoli-Libya

[4] Rana S, Mehra R. Design &simulation of RISC processor using hyper pipelining technique. IOSR Journal of Mechanical and Civil Engineering (IOSR-JMCE). 2013;9(2):49-57.

[5] Trivedi P. Design &analysis of 16-bit RISC processor using low power pipelining. 2015 International Conference on Computing, Communication & Automation (ICCCA); 2015:1294-7.

[6] Finlayson I, Uh GR, Whalley DB, Tyson G. An overview of static pipelining. IEEE Computer Architecture Letters. 2012; 11(1):17-20.

[7] Cheah HY, Fahmy SA, Kapre N. Analysis and optimization of a deeply pipelined FPGA soft processor. 2014 International Conference on Field-Programmable Technology (FPT); 2015. p. 235-8.

[8] Hoganson KE. High-performance computer architecture and algorithm simulator Journal on Educational Resources in Computing. 2002; 2(1):131-48.

[9] Grunbacher H. Teaching computer architecture/organisation using simulators. 28th Annual Frontiers in Education Conference, FIE'98. Treitlstrasse Vienna Austria. 1998; 3:1107-12.

[10]Osée M, Richard A, Biest AV, Mathys P. Educational simulation of the RiSC processor. International Conference on Engineering Education(ICEE 2007); 2007.

[11]Hoganson K. The unified parallel speedup model and simulator Southeast Regional ACM Conference; 2001. p. 1-23.

[12]DE2i-150 FPGA System Manual, ALTERA, 2013.