

REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING



CPU DESIGN WITH FPGA

15011901 – Natiq AGHAYEV

SENIOR PROJECT

Advisor
Asst. Prof. Erkan USLU

July, 2020

ACKNOWLEDGEMENTS

This project was something I wanted to do for a long time. First of all, I would like to extend my thanks to Res. Asst. Alper EGITMEN who supported me spiritually throughout the project. I would like to thank Asst. Prof. Erkan USLU who provided every environment, provided all kinds of hardware assistance and made an important contribution to the shaping of the project. Finally, I would like to thank my dear family, who always supported me.

Natiq AGHAYEV

TABLE OF CONTENTS

LIST OF ABBREVIATIONS	v
LIST OF FIGURES	vi
ABSTRACT	vii
ÖZET	viii
1 Introduction	1
2 Literature Review	2
3 Feasibility	3
3.1 Technical Feasibility	3
3.1.1 Software Feasibility	3
3.1.2 Hardware Feasibility	4
3.2 Labor Force and Time Planning	4
3.3 Legal Feasibility	4
3.4 Economic Feasibility	5
4 System Analysis	6
5 System Design	7
5.1 The Instruction Set	7
5.1.1 The Instruction Set	7
5.1.2 Operation Codes of Instructions	10
5.1.3 Operation list	12
5.1.4 Instruction types	12
5.2 Components of a Microprocessor	13
5.2.1 Register File Components	13
5.2.2 Arithmetic Logic Unit	14
5.2.3 Control Signal Generator	14
5.2.4 Memory Address Creator	14
5.2.5 Instruction Decoder	14

5.2.6	Operation Code Creator	14
5.3	Memory Segmentation	14
5.4	Compiler Spesific Commands	14
6	Software	16
6.1	Running steps	16
7	Experimental Results	20
8	Results	21
	References	22
	Curriculum Vitae	23

LIST OF ABBREVIATIONS

FPGA	Field-programmable Gate Array
CPU	Central processing unit
SoC	System-on-Chip
RAM	Random Access Memory
ROM	Read-Only Memory
RISC	Reduced Instruction Set Computer
GPU	Graphics Precessing Unit
ASIC	Application Specific Integrated Circuit
FSM	Finite State Machine

LIST OF FIGURES

Figure 3.1	Gannt Diagram	4
Figure 6.1	Compiling code	16
Figure 6.2	ModelSim ALTERA STARTER EDITION	17
Figure 6.3	Adding verilog files	17
Figure 6.4	Adding header files	18
Figure 6.5	Starting simulation	18
Figure 6.6	Selecting simulation	19
Figure 6.7	Running simulation	19

CPU Design with FPGA

Natiq AGHAYEV

Department of Computer Engineering
Senior Project

Advisor: Asst. Prof. Erkan USLU

In this project, it is thought to design a microprocessor that uses RISC microprocessor architecture, which has operations in a single cycle, with register file, ALU, control unit and other units, and it is considered to use FPGA to implement this project. Project steps can be listed as designing ISA architecture, coding hardware modules, designing and coding compiler for assembly language, and performing tests.

Keywords: ISA Architecture, FPGA, Verilog, Python, ALU, Microprocessor Design, RISC

FPGA ile CPU Tasarımı

Natiq AGHAYEV

Bilgisayar Mühendisliği Bölümü
Bitirme Projesi

Danışman: Dr.Öğr.Üyesi Erkan USLU

Bu projede RISC microprocessor architecture kullanılarak tek çevrimde işlemleri yapan, register file, ALU, kontrol birimi ve başka birimlerin olduğu mikroişlemci tasarlanması düşünülmüştür ve bu projenin gerçekleştirilmesi için FPGA kullanılması düşünülmüştür. Proje adımları ISA mimarisinin tasarlanması, donanımsal modüllerin kodlanması, assembly dili için derleyici tasarlanması ve kodlanması, testlerin yapılması olarak sıralanabilir.

Anahtar Kelimeler: ISA Mimarisi, FPGA, Verilog, Python, ALU, Mikroişlemci Tasarımı

1

Introduction

Today, computer systems attract attention in many areas. Computers are used for different purposes in airports, public areas, schools, workplaces and homes. Compared to their intended use, the capacity and speed of computers vary. In general, although software is more prominent, hardware problems can sometimes disrupt or stop the workflow on computers.

In fact, every software project can be implemented hardware. However, given the size and requirements of the projects, hardware improvements are often put on the second plan as they can further extend the project construction time. Hardware projects are mostly carried out with the aim of faultless and convenient operation of the basic building blocks of the designed system.

Microprocessors are developed to meet these requirements. Instead of designing a special microprocessor for each computer, standards have been determined and microprocessors have been produced compared to these standards. And during these production processes, In the past, field programmable gate arrays (FPGAs) have been used to absorb glue logic, perform signal processing, and even to prototype system-on-chip (SoC) ASICs. Now with the advent of large, fast, cheap FPGAs, such as Xilinx Spartan-II (100,000 1 ns “gates” for \$15, quantity one), it is practical and cost-effective to skip the ASIC and ship volume embedded systems in a single FPGA plus off-chip RAM and ROM – the FPGA implements all of the system logic including a processor core. [1]

Within the scope of this project, it is aimed to develop single cycle CPU design and compatible assembler based on RISC architecture by using Altera DE2.

2 Literature Review

Known for their flexibility, Field Programmable Gate Arrays (FPGA) are widely used for ASIC emulation, glue-logic consolidation, or as a solution for applications with high volatility. FPGAs facilitate quick time to market, and their incredible power of re-programmability often makes them the heart of a system [2]. Designers of embedded and real-time systems are continually challenged to meet tighter system requirements at better price-performance ratios [3]. The combining of one or more CPU's and an FPGA fabric on the same die is growing in popularity. Such programmable system-on-chip (PSOC) systems promise performance and development time advantages over conventional technology[4]. CPU-FPGA heterogeneous acceleration platforms have shown great potential for continued performance and energy efficiency improvement for modern data centers, and have captured great attention from both academia and industry. The increasing demand for energy-efficient high-performance computing has stimulated a growing number of heterogeneous architectures that feature hard-ware accelerators or coprocessors, such as GPUs (Graphics Processing Units), FPGAs (Field Programmable Gate Arrays), and ASICs (Application Specific Integrated Circuits) [5]. The addition of system instrumentation features have been sporadically incorporated into processor architectures over the last several decades. Particular emphasis areas of high performance, embedded and real-time computing are reviewed in terms of software and hardware measurements and approaches representing active research directions [6]. The design of digital electronic circuits has entered a new era, which make it possible for designers to customize their own special CPU. So it is a great significance to design an economic and applicable CPU core based on FPGA for the cost reduction and the possession of intellectual property [7].

3.1 Technical Feasibility

3.1.1 Software Feasibility

3.1.1.1 Programming Language : Verilog

Verilog is a Hardware Description Language; a textual format for describing electronic circuits and systems. Applied to electronic design, Verilog is intended to be used for verification through simulation, for timing analysis, for test analysis (testability analysis and fault grading) and for logic synthesis. VHDL is more verbose than Verilog and it also has a non-C like syntax. With VHDL, you have a higher chance of writing more lines of code. That's why Verilog was used in this project instead of VHDL.

3.1.1.2 Programming Language : Python

Python commands were used to convert binary codes. It has added speed to the project in terms of its positive aspects such as ease of coding and reducing coding time.

3.1.1.3 Software Development Environment : ModelSim ALTERA STARTER EDITION 10.1d

ModelSim ALTERA STARTER EDITION 10.1d is used to run and analyze hardware modules and test modules programmed in one of the hardware description languages.

3.1.1.4 Software Development Environment : Quartus II 13.0sp1 (64-bit) Web Edition

The Quartus II Web Edition FPGA design software includes everything that the user needs to design for the Altera FPGA families.

3.1.1.5 Operating System : Windows 10

Since the project was developed in the laboratory within the university, the operating system installed on the computers in the laboratory was used. Since there is no special need for a special operating system in the project, Windows 10 installed on computers was used as the operating system.

3.1.2 Hardware Feasibility

VGA screen, FPGA board, PS2 keyboard are the top priorities for the realization of the project. 11GB disk memory is required for program installation. Windows 7 or higher 64-bit operating system is required as a minimum requirement for the installation of the program. Windows 7 operating system requires 20 GB available hard disk space, 1 gigahertz (GHz) processor, DirectX 9 graphics device with WDDM 1.0 or higher driver. Depending on resolution, video playback may require additional memory and advanced graphics hardware. Product functionality and graphics may vary based on your system configuration. Some features may require advanced or additional hardware. Windows 7 was designed to work with today's multi-core processors. All 32-bit versions of Windows 7 can support up to 32 processor cores, while 64-bit versions can support up to 256 processor cores.

3.2 Labor Force and Time Planning

The time plan of the project is shown as a Gantt diagram at the below.

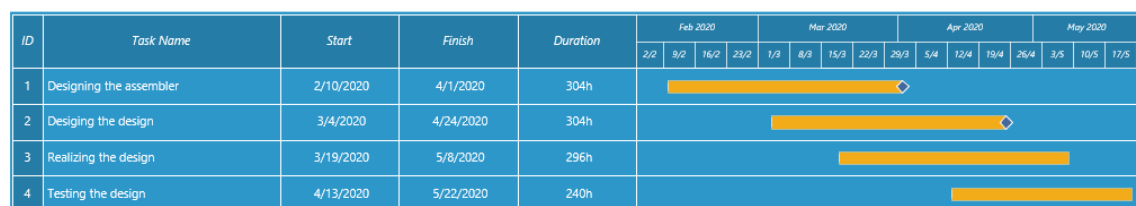


Figure 3.1 Gantt Diagram

3.3 Legal Feasibility

Windows 10 Education is designed for students, workplaces and can be downloaded at no cost. The free versions of Quartus II 13.0sp1 (64-bit) Web Edition and ModelSim ALTERA STARTER EDITION are used.

3.4 Economic Feasibility

The price of the FPGA board is 1100 TL, the price of the PS2 keyboard is 200 TL, the price of the VGA screen is 490 TL, the price of the VGA cable is 35 TL. Total amount is determined as 1825 TL.

4

System Analysis

Based on the RISC architecture, Altera DE2 FPGA is used. It is planned to execute commands in one cycle. Along with the CPU, program memory and data memory are also created. 16 16-bit registers were created in the CPU and 8 of these registers are divided to low and high bytes.

The instruction set should include simple arithmetic operations (addition, increment, subtraction, ...), simple logic operations (and, or, not, ...), translation / shift, data transfer, conditional / unconditional branching commands. Compatible assembler has been developed.

It is planned to provide output via terminal application with VGA controller on DE2, keyboard input with PS2 via FPGA. Verilog programming language is used in CPU design.

5

System Design

5.1 The Instruction Set

5.1.1 The Instruction Set

While designing the instruction set, it was inspired by the instruction set of the x 8086 microprocessor. From the values in the Table 5.1, r indicates that the command affects the flag, x indicates that the command does not affect the flag, 0 indicates that the commands set the flag value to 0.

COMMAND	OPCODE (6 bit)	OPERAND1	OPERAND2	EXPLANATION	CF	PF	OF	SF	ZF
ADD	000000	<reg1> (5bit)	<reg2> (5bit)	$reg2 = reg1 + reg2$	r	r	r	r	r
ADC	000001	<reg1> (5bit)	<reg2> (5bit)	$reg2 = reg1 + reg2 + CF$	r	r	r	r	r
SUB	000010	<reg1> (5bit)	<reg2> (5bit)	$reg2 = reg1 - reg2$	r	r	r	r	r
AND	000011	<reg1> (5bit)	<reg2> (5bit)	$reg2 = reg1 \& reg2$	0	r	0	r	r
OR	000100	<reg1> (5bit)	<reg2> (5bit)	$reg2 = reg1 reg2$	0	r	0	r	r
XOR	000101	<reg1> (5bit)	<reg2> (5bit)	$reg2 = reg1 \wedge reg2$	0	r	0	r	r
INC	000110	<reg1> (5bit)		$reg1 = reg1 + 1$	x	r	r	r	r
DEC	000111	<reg1> (5bit)		$reg1 = reg1 - 1$	x	r	r	r	r
NEG	001000	<reg1> (5bit)		$reg1 = (-1) * reg1$	r	r	r	r	r
NOT	001001	<reg1> (5bit)		reverse of all bits of reg1	x	x	x	x	x
SHRA	001010	<reg1> (5bit)		shift right(signed), sign don't care	r	x	r	x	x
SHRS	001011	<reg1> (5bit)		shift right(unsigned), sign care	r	x	0	x	x
SHLA	001100	<reg1> (5bit)		shift left(signed), sign don't care	r	x	r	x	x
SHLS	001101	<reg1> (5bit)		shift left(unsigned), sign care	r	x	0	x	x
ROR	001110	<reg1> (5bit)		rotate right without carry	r	x	r	x	x
ROL	001111	<reg1> (5bit)		rotate left without carry	r	x	r	x	x
RCR	010000	<reg1> (5bit)		rotate right with carry	r	x	r	x	x
RCL	010001	<reg1> (5bit)		rotate left with carry	r	x	r	x	x
TEST	010010	<reg1> (5bit)	<reg2> (5bit)	$reg1 \& reg2$	0	r	0	r	r

COMMAND	OPCODE (6 bit)	OPERAND1	OPERAND2	EXPLANATION	CF	PF	OF	SF	ZF
JA	010011	<reg> (5bit)		if reg1>reg2(unsigned) (given in CMP command)	x	x	x	x	x
JNBE	010011	<reg> (5bit)		if reg1>reg2(unsigned) (given in CMP command)	x	x	x	x	x
JB	010100	<reg> (5bit)		if reg1<reg2(unsigned) (given in CMP command)	x	x	x	x	x
JNAE	010100	<reg> (5bit)		if reg1<reg2(unsigned) (given in CMP command)	x	x	x	x	x
JC	010100	<reg> (5bit)		if CF==1	x	x	x	x	x
JNB	010101	<reg> (5bit)		if reg1>=reg2(unsigned) (given in CMP command)	x	x	x	x	x
JAE	010101	<reg> (5bit)		if reg1>=reg2(unsigned) (given in CMP command)	x	x	x	x	x
JNC	010101	<reg> (5bit)		if CF==0	x	x	x	x	x
JBE	010110	<reg> (5bit)		if reg1<=reg2(unsigned) (given in CMP command)	x	x	x	x	x
JNA	010110	<reg> (5bit)		if reg1<=reg2(unsigned) (given in CMP command)	x	x	x	x	x
JL	010111	<reg> (5bit)		if reg1<reg2(signed) (given in CMP command)	x	x	x	x	x
JNGE	010111	<reg> (5bit)		if reg1<reg2(signed) (given in CMP command)	x	x	x	x	x
JNL	011000	<reg> (5bit)		if reg1>=reg2(signed) (given in CMP command)	x	x	x	x	x
JGE	011000	<reg> (5bit)		if reg1>=reg2(signed) (given in CMP command)	x	x	x	x	x
JLE	011001	<reg> (5bit)		if reg1<=reg2(signed) (given in CMP command)	x	x	x	x	x
JNG	011001	<reg> (5bit)		if reg1<=reg2(signed) (given in CMP command)	x	x	x	x	x
JNLE	011010	<reg> (5bit)		if reg1>reg2(signed) (given in CMP command)	x	x	x	x	x
JG	011010	<reg> (5bit)		if reg1>reg2(signed) (given in CMP command)	x	x	x	x	x
JE	011011	<reg> (5bit)		if reg1==reg2 (given in CMP command)	x	x	x	x	x

COMMAND	OPCODE (6 bit)	OPERAND1	OPERAND2	EXPLANATION	CF	PF	OF	SF	ZF
JZ	011011	<reg>(5bit)		if ZF==1	x	x	x	x	x
JNE	011100	<reg>(5bit)		if reg1!=reg2(given in CMP command)	x	x	x	x	x
JNZ	011100	<reg>(5bit)		if ZF==0	x	x	x	x	x
JO	011101	<reg>(5bit)		if OF==1	x	x	x	x	x
JNO	011110	<reg>(5bit)		if OF==0	x	x	x	x	x
JS	011111	<reg>(5bit)		if SF==1	x	x	x	x	x
JNS	100000	<reg>(5bit)		if SF==0	x	x	x	x	x
JP	100001	<reg>(5bit)		if PF==1	x	x	x	x	x
JPE	100001	<reg>(5bit)		if PF==1	x	x	x	x	x
JNP	100010	<reg>(5bit)		if PF==0	x	x	x	x	x
JPO	100010	<reg>(5bit)		if PF==0	x	x	x	x	x
JMP	100011	<reg>(5bit)		go to address in <reg>	x	x	x	x	x
CMP	100100	<reg1>(5bit)	<reg2>(5bit)	comparing reg1 and reg2 (flags are affected)	r	r	r	r	r
SETC	100101			CF=1	1	x	x	x	x
CLC	100110			CF=0	0	x	x	x	x
LDIL	100111	<constant value>(8bit)		low 8 bits of <constant value> saved in RLI[7:0] reg.	x	x	x	x	x
LDIH	101000	<constant value>(8bit)		high 8 bits of <constant value> saved in RLI[15:8] reg.	x	x	x	x	x
SETADDL	101001	<variable>or <label>(8bit)		low 8 bits of <variable>or <label> address saved in RADR[7:0] reg.	x	x	x	x	x
SETADDH	101010	<variable>or <label>(8bit)		high 8 bits of <variable>or <label> address saved in RADR[15:8] reg.	x	x	x	x	x
GETDATA	101011	<reg1>(5bit)	<reg2>(5bit)	write 16 bit data from <reg1>address in DS to <reg2> (reg2 must be 16bit)	x	x	x	x	x
SETDATA	101110	<reg1>(5bit)	<reg2>(5bit)	write 16 bit data from <reg1>to <reg2>address.	x	x	x	x	x

COMMAND	OPCODE (6 bit)	OPERAND1	OPERAND2	EXPLANATION	CF	PF	OF	SF	ZF
MOV	101111	<reg1>(5bit)	<reg2>(5bit)	<reg2>=<reg1> (if reg1 is 16 and reg2 is 8 bit then low 8 bit of reg1 will be assigned to reg2.)	x	x	x	x	x
PUSH	110010	<reg1>(5bit)		push <reg>data to stack	x	x	x	x	x
POP	110011	<reg1>(5bit)		pop data from stack to <reg>	x	x	x	x	x
GETSDATA	110100	<reg1>(5bit)	<reg2>(5bit)	<reg2>= data at <reg1>address in stack	x	x	x	x	x
HALT	110111			STOP THE PC	x	x	x	x	x

Table 5.1 Instruction Set

5.1.2 Operation Codes of Instructions

Each command has its own list of operations. After reading instruction, operations are created by Operation Code Creator. Microprocessor executes instructions as FSM and waits for next clock cycle.

COMMANDS	operation1	operation2	operation3	operation4
ADD	0	1	2	10
ADC	0	1	2	10
SUB	0	1	2	10
AND	0	1	2	10
OR	0	1	2	10
XOR	0	1	2	10
INC	0	1	2	10
DEC	0	1	2	10
NEG	0	1	2	10
NOT	0	1	2	10
SHRA	0	1	2	10
SHRS	0	1	2	10
SHLA	0	1	2	10
SHLS	0	1	2	10
ROR	0	1	2	10
ROL	0	1	2	10

COMMANDS	operation1	operation2	operation3	operation4
RCR	0	1	2	10
RCL	0	1	2	10
TEST	0	1	10	
JA	0	1	8	10
JNBE	0	1	8	10
JB	0	1	8	10
JNAE	0	1	8	10
JC	0	1	8	10
JNB	0	1	8	10
JAE	0	1	8	10
JNC	0	1	8	10
JBE	0	1	8	10
JNA	0	1	8	10
JL	0	1	8	10
JNGE	0	1	8	10
JNL	0	1	8	10
JGE	0	1	8	10
JLE	0	1	8	10
JNG	0	1	8	10
JNLE	0	1	8	10
JG	0	1	8	10
JE	0	1	8	10
JZ	0	1	8	10
JNE	0	1	8	10
JNZ	0	1	8	10
JO	0	1	8	10
JNO	0	1	8	10
JS	0	1	8	10
JNS	0	1	8	10
JP	0	1	8	10
JPE	0	1	8	10
JNP	0	1	8	10
JPO	0	1	8	10
JMP	0	1	8	10
CMP	0	1	10	
SETC	1	10		

COMMANDS	operation1	operation2	operation3	operation4
CLC	1	10		
LDIL	2	10		
LDIH	2	10		
SETADDL	2	10		
SETADDH	2	10		
GETDATA	0	3	2	10
SETDATA	0	4	10	
MOV	0	2	10	
PUSH	0	5	10	
POP	0	6	2	10
HALT	9			

Table 5.2 Operation codes of commands

5.1.3 Operation list

Operation	Code
Read data from register file	0
Enable alu and run instruction	1
Write data to register file	2
Read data from memory	3
Write data to memory	4
Push data to stack	5
Pop data from stack	6
Set program counter	8
Halt	9
Read instruction	10

Table 5.3 Operations list

5.1.4 Instuction types

There are 4 instruction types: OPR1R2, OPR1, OPD8, OP

1. OPR1R2

OPR1R2 type instructions require 2 registers as operands.

2. OPR1

OPR1 type instructions require 1 register as operand.

3. OPD8

OPD8 type instructions require 8 bit data as operand.

4. OP

OP type instructions doesn't require anything as operand.

5.2 Components of a Microprocessor

5.2.1 Register File Components

5.2.1.1 Register File

There are 16 16-bit registers and 8 of these registers are divided to low and high bytes. AX1, BX1, CX1, DX1, AX2, BX2, CX2, DX2, R1, R2, R3, R4, SP, BP, RLI, RADR are 16-bit registers and AX1, BX1, CX1, DX1, AX2, BX2, CX2, DX2 can be divided to low and high bytes. AL1, BL1, CL1, DL1, AH1, BH1, CH1, DH1, AL2, BL2, CL2, DL2, AH2, BH2, CH2, DH2 ARE 8-BIT registers. AL1 is low byte of AX1 and AH1 is high byte of AX1. RLI register is used for loading constant and RADR register is used for loading address. The register file is used for either reading or writing. In register file 2 read can be at 4-bit register address, w_lb, w_hb signals comes from Register File Address Creator and is used for selecting register. The register file supports reading from 2 addresses at the same time and writing to 1 address

5.2.1.2 Register File Address Creator

All registers can be coded with 5-bit. Register File Address Creator gets 5-bit register code and Instruction register(IR) as input and generate 4-bit register address, w_lb, w_hb signals. w_lb signal is used for selecting low byte of register and w_hb signal is used for selecting low byte of register. if entered instruction is LDIL or LDIH or SETADDL or SETADDH then w_lb and w_hb signals is set or reset . Other instructions doesn't affect w_lb and w_hb signals. In this situation, w_lb and w_hb is set or reset according to 5-bit register address.

5.2.1.3 Register Address Setter

When the LDIL and LDIH commands are executed, the RLI register address is considered to be a 4-bit register address. When the SETADDL and SETADDH commands are executed, the RADR register address is considered to be a 4-bit register address. When the PUSH and POP commands are executed, the SP register address is considered to be a 4-bit register address and one of these registers is used as output of this module.

5.2.2 Arithmetic Logic Unit

Arithmetic Logic Unit performs arithmetic and bitwise operations on only 16 bit integer binary numbers and contains Zero flag, Sign flag, Overflow flag, Carry flag, Parity flag. Branching instructions are executed in ALU.

5.2.3 Control Signal Generator

This component generates signal for each entered operation. This signals enable or disable register file and arithmetic logic unit, give permission to write or read register file or memory.

5.2.4 Memory Address Creator

This component generates 18-bit memory address from IR and entered 16-bit address.

5.2.5 Instruction Decoder

This component generates register addresses, 8-bit data and IR from entered binary code.

5.2.6 Operation Code Creator

This component generates operations from entered IR.

5.3 Memory Segmentation

The Memory Segmentation is the same as the memory segmentation of the 8086 processor. There are 3 segments: Code segment, Data segment, Stack segment. Each of segments has 64kB memory and can be addressed with 16-bit registers.

5.4 Compiler Specific Commands

1. ASSIGN

ASSIGN command used for assigning shadow names for registers. Sometimes using registers can be confusing. And naming them with understandable names can make programs more readable.

Usage : ASSIGN "shadow_name", <reg>

2. DEASSIGN DEASSIGN command is for removing link between shadow_name and register.

Usage : DEASSIGN "shadow_name"

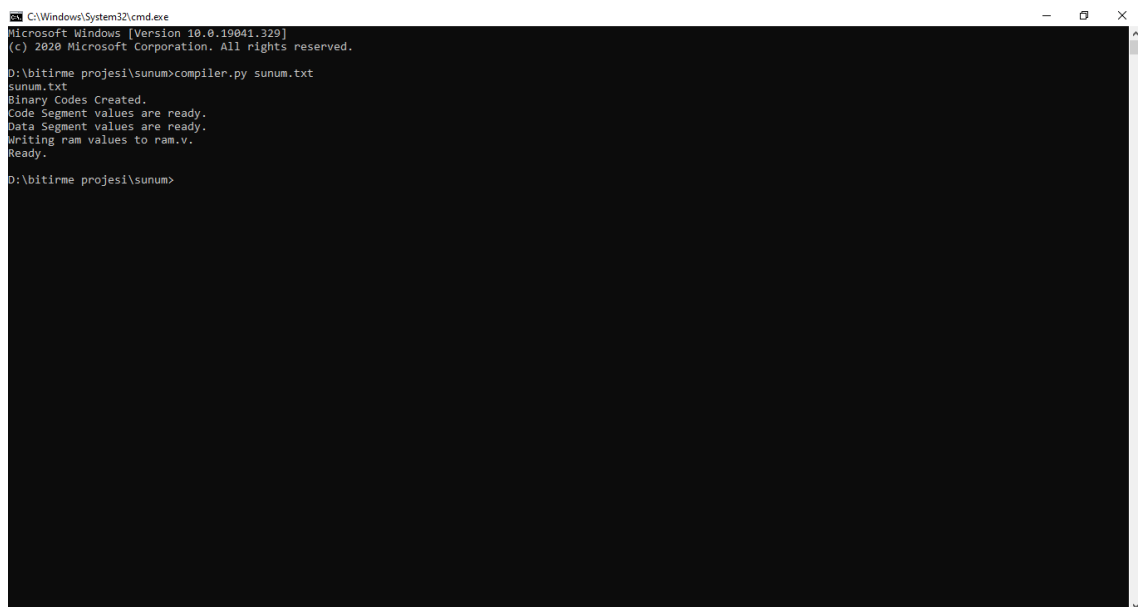
3. DFN DFN command is used for initializing variables.

Usage: DFN Var_name Number(s)

Within the scope of the project, the piece of code designed for each algorithm is converted to opcodes corresponding to the commands in the instruction set and the instructions executed on ModelSim ALTERA STARTER EDITION. After each command, related ones change from register values and ram memory.

6.1 Running steps

1. Firstly, instructions must be converted to binary code. After converting instructions to binary code, output file(ram.v) must be placed in project folder.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19041.329]
(c) 2020 Microsoft Corporation. All rights reserved.

D:\bitirme\projesi\sunum>compiler.py sunum.txt
sunum.txt
Binary Codes Created.
Code Segment values are ready.
Data Segment values are ready.
Writing ram values to ram.v.
Ready.

D:\bitirme\projesi\sunum>
```

Figure 6.1 Compiling code

2. After placing output file to project folder, Modelsim must be opened.
3. After opening Modelsim, verilog files must be added to Modelsim.

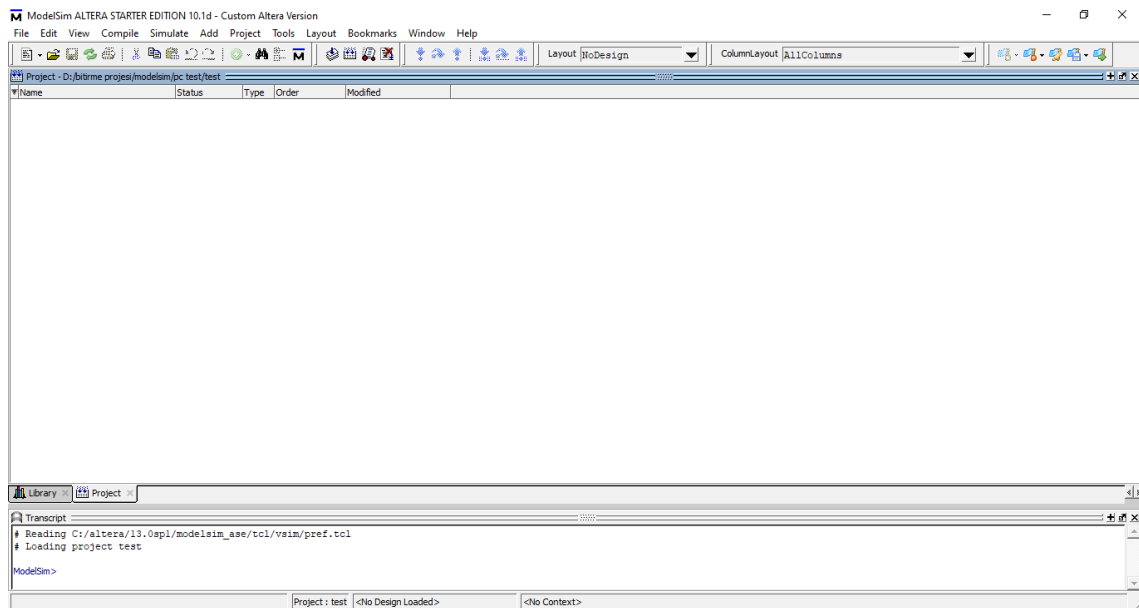


Figure 6.2 ModelSim ALTERA STARTER EDITION

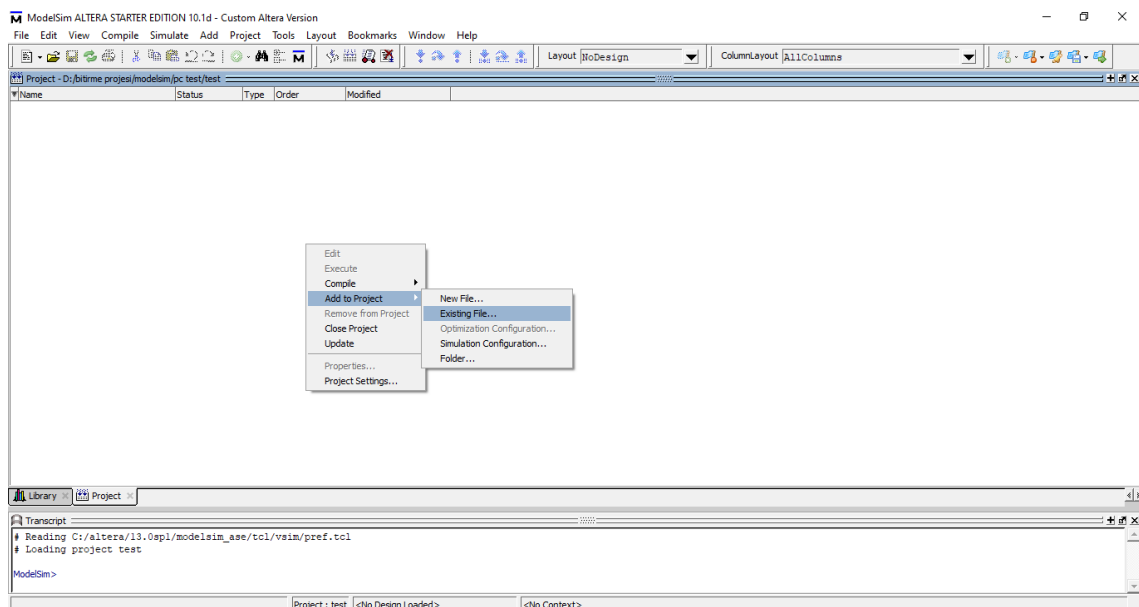


Figure 6.3 Adding verilog files

4. After adding files, header files must be added. For adding header files, Right Click->Properties->VerilogSystemVerilog->Include Directory path must be followed and header files must be added.
5. After adding all files, simulation can be started. Simulation->Start Simulation path must be followed for starting simulation.
6. After starting simulation, simulation name must be selected.
7. For running simulation Run-all must be clicked.

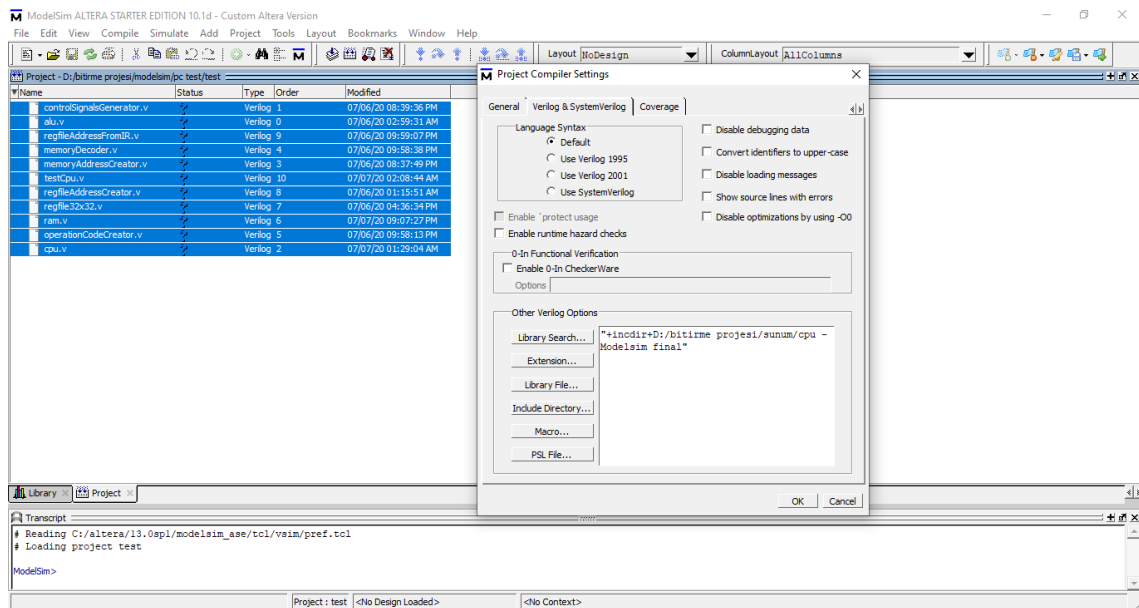


Figure 6.4 Adding header files

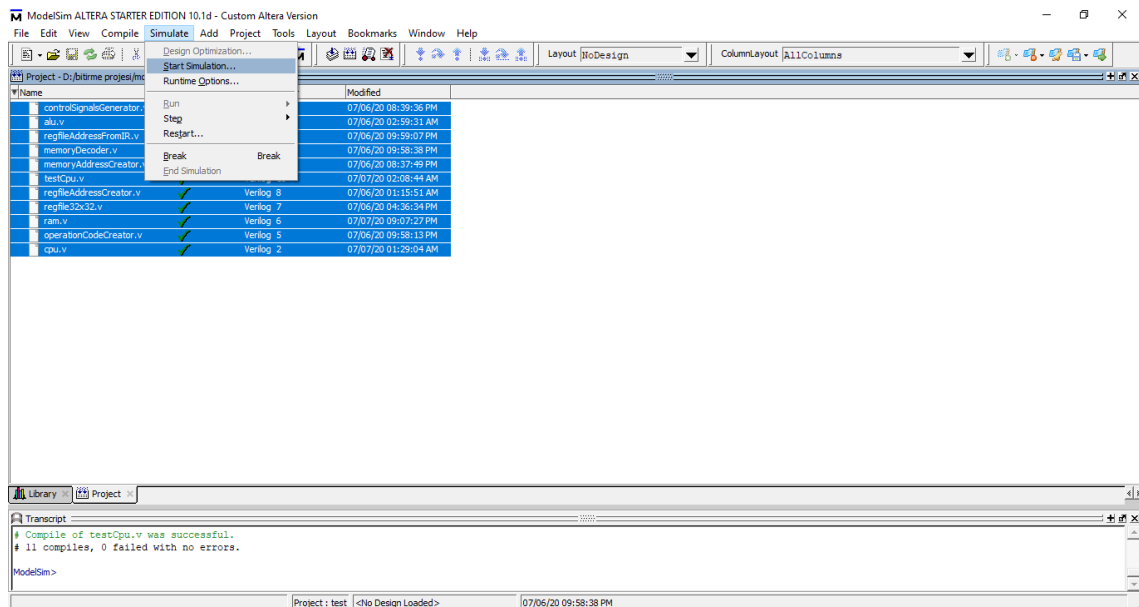


Figure 6.5 Starting simulation

After running simulation, ram values can be seen in Objects section.

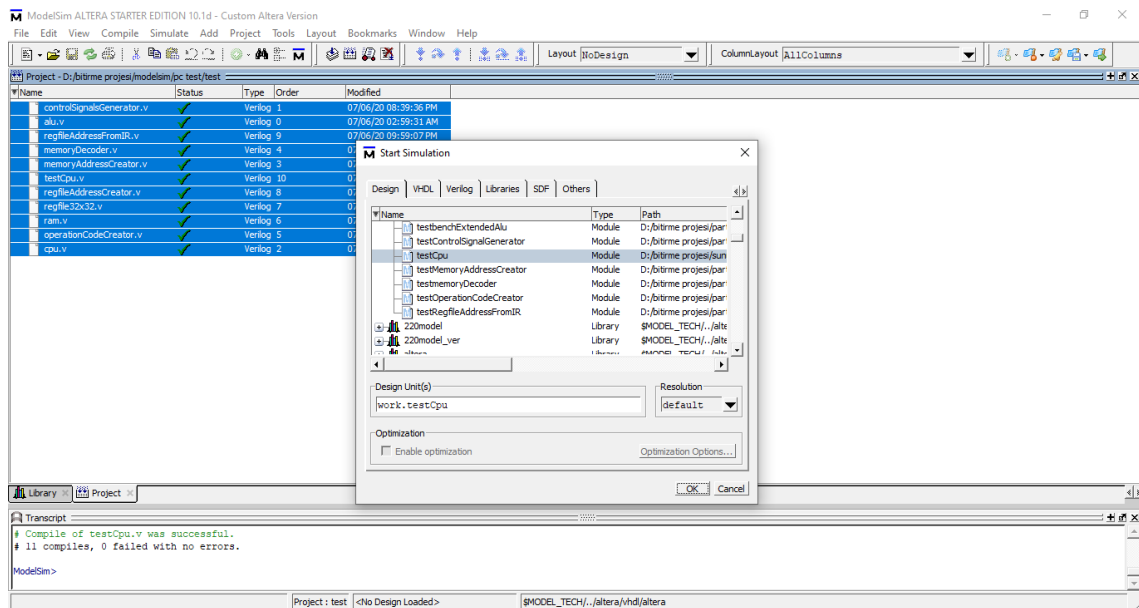


Figure 6.6 Selecting simulation

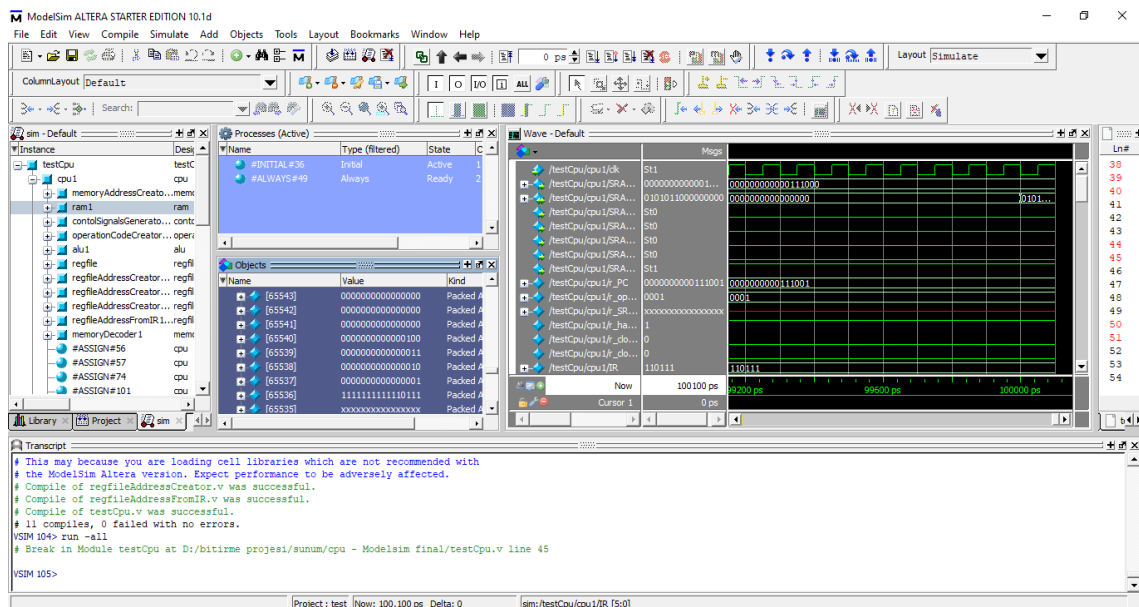


Figure 6.7 Running simulation

7

Experimental Results

None of the programs were run on FPGA. Compared to the values observed in simulation, all commands run in one cycle. The codes could not be converted to the appropriate format for FPGA because the timing analysis of the codes could not be done very well and the verilog programming language was not very well-known. All results are simulation-based. Since the FSM structure is used in the microprocessor design and the hardware models used are independent of the synchronizing clock, a sudden response is received in every clock pulse. Since the ram structure planned to be used is sram, reading binary codes, writing data to memory and reading data from memory is completed in a very short time. Although there is no access to the real ram, a ram module similar to the sram structure is programmed and the data is read from within this module. Bubble-sort sorting algorithm was run with the designed assembly language and the results were observed to be correct.

8 Results

In the initial design phase, it was not possible to execute commands in each cycle, since there were no operation codes for each command. However, after adding the operation codes and applying the FSM structure, it was possible to execute the codes in a single cycle. When compared with CISC architecture, it is necessary to design more advanced hardware that will perform operations that can work longer in RISC architecture. In other words, a processor designed in CISC architecture consists of very small modules and runs a large process piece by piece. In order to do the same operation in RISC architecture, we need to either design advanced hardware pieces, for example, the displacement circuit embedded in the register file, or use operation codes. When operation codes are used, it is necessary to determine the delays between the modules. If advanced modules are not designed as mentioned above, this will manifest itself in prolonging coding times and doing what is desired with more than one command, and this can be overcome by preparing a specific templates for the steps used in general programming logic.

References

- [1] J. Gray, “Designing a simple fpga-optimized risc cpu and system-on-a-chip,” Jan. 2002.
- [2] J. D. Luker and V. B. Prasad, “Risc system design in an fpga,” in *Proceedings of the 44th IEEE 2001 Midwest Symposium on Circuits and Systems. MWSCAS 2001 (Cat. No.01CH37257)*, vol. 2, 2001, 532–536 vol.2.
- [3] D. Andrews, D. Niehaus, and P. Ashenden, “Programming models for hybrid cpu/fpga chips,” *Computer*, vol. 37, no. 1, pp. 118–120, 2004.
- [4] E. Roesler and B. Nelson, “Debug methods for hybrid cpu/fpga systems,” in *2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings.*, 2002, pp. 243–250.
- [5] Y.-k. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, “A quantitative analysis on microarchitectures of modern cpu-fpga platforms,” in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16, Austin, Texas: Association for Computing Machinery, 2016, ISBN: 9781450342360. DOI: 10 . 1145/2897937 . 2897972. [Online]. Available: <https://doi.org/10.1145/2897937.2897972>.
- [6] R. Fryer, “Fpga based cpu instrumentation for hard real-time embedded system testing,” *SIGBED Rev.*, vol. 2, no. 2, pp. 39–42, Apr. 2005. DOI: 10 . 1145 / 1121788 . 1121800. [Online]. Available: <https://doi.org/10.1145/1121788.1121800>.
- [7] Y. Zhang and L. Bao, “The design of an 8-bit cisc cpu based on fpga,” in *2011 7th International Conference on Wireless Communications, Networking and Mobile Computing*, 2011, pp. 1–4.

Curriculum Vitae

FIRST MEMBER

Name-Surname: Natiq AGHAYEV

Birthdate and Place of Birth: 16.01.1998, Azerbaijan

E-mail: aganatiq98@gmail.com

Phone: 05525303169

Practical Training: Yıldız Teknik Üniversitesi, Akıllı Sistemler Laboratuvarı

Project System Informations

System and Software: Windows İşletim Sistemi, Verilog, Python3

Required RAM: 2GB

Required Disk: 256MB