

CE 6302.001 - Microprocessor Systems

Design and Implementation of Single Cycle Computer using Verilog HDL

Professor: Dr. Terrell Bennett

**Name**: VIVEK NAGALAPURA RAVINDRA

**Net ID:** vxn170230

**Date**: 6th December 2019

# Introduction

The objective of this project is to design and implement a single cycle computer in using Xilinx ISE and implement it using Xilinx FPGA board available in the Lab. The Data Width is 16 bit and Instruction Width is 16 bit. The following instructions are supported:

* Register Arithmetic Instructions – ADD, SUB
* Immediate Arithmetic Instruction – ADDI, SUBI
* Load Instruction - LOAD
* Store Instruction - STORE
* Branch Instruction – BNQ

Steps followed in completion of the project:

* Understand the working of single cycle processor data path – Register File, Instruction Memory, Data Memory, ALU and Instruction decode, control logic.
* Write Verilog modules and testbench for each of the components listed above using Xilinx ISE.
* Perform Simulation of the each of the modules and the top-level module using the Xilinx ISIM simulator.
* Synthesize the Verilog code and implement it on Xilinx FPGA board available in the Lab.

# Block Diagram

Switches

ALU

Instruction Memory

Data Memory

Register File

S

LEDs

S

S

S

S

Program Counter

S

S

S

Clock – 100 MHz

# Environment

The following environment to implement the single cycle computer:

* Xilinx ISE 14.7 is installed on Windows 10 (Installs virtual machine with a Shared Directory)
* Internal Simulation ISIM is used for Simulation and Verification
* Xilinx Synthesis Tool is used to generate the programming file (BIT FILE)
* Digilent Adept NEXYS3 Tool is used the program the Xilinx Spartan 6 FPGA board available in the Lab.
* The 8 Built-in LEDs are used to view the hexadecimal number which is Nth Fibonacci Number.

# RTL Schematic

A screenshot of a computer

Description automatically generated

# Working

The following Verilog modules are included:

1. Processor – Instantiates all other modules. This is the Top Module in the design. 100 MHz clock clk and [7:0] sw are used as input and [7:0] Led are used as output. These are also specified in the UCF file.
2. Inst\_Mem – Holds the machine code for each 16 bit instruction.
3. BranchUnit – Increments or directs the Program counter which points to the next instruction
4. Control – Decodes the instructions and outputs the required control signals to perform operations to move the data along the Data Path.
5. RegFile – This is a memory module for the 16 bit registers in the Processor.
6. DataMemory – Data Memory within the Processor.
7. ALU – (16 bit) Performs the operations specified by the instruction.
8. Start – Initializes the Program counter to 0 to which executes the instructions.
9. SignExtend – Extends the sign of the data from 6 bits to 16 bit which is the Data Width so that the data can be fed into the ALU for arithmetic operations.
10. Mux – Many Mux are used with the control signals to control the data path for each instruction.
11. Test benches – Test benches are used for Simulation of the design.

# Control Signals

Depending on the instruction, the required control signals are sent so that the correct data travels through the Data Path. The following table shows the control signals which are generated for each instruction:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction | Reg  Dst | ALUsrc | Memto  Reg | Reg  Write | MemRead | Mem  Write | Branch | ALUop |
| ADD | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 000 |
| SUB | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 001 |
| LOAD | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 000 |
| STORE | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 000 |
| ADDI | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 000 |
| SUBI | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 001 |
| BNQ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 001 |

* *RegDst* is used for the 3 register instruction.
* *ALUsrc* is used to select Data from the register or immediate value.
* *MemtoReg* is used in the Load instruction to load the register with the data present in the data memory.
* *RegWrite* is used to write a value to the Register in the Register File.
* *MemRead* and *MemWrite* are used for Load and Store instructions respectively. *Branch* is used in the BNQ instruction.
* *ALUop* is used to select the ALU operation – either add or subtract.

# Instruction Formats Supported (All are 16 bit instructions)

ADD and SUB Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode[15:13] | Src Reg[12:10] | Src Reg[9:7] | Dst Reg[6:4] | Not Used[3:0] |

LOAD, STORE, BNQ, ADDI and SUBI Instructions Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode[15:13] | Src Reg[12:10] | Dst Reg[9:7] | Immediate Data [6:0] |

# Assembly Program to find the Nth Fibonacci Number

Register R3 has the value N. Register R4 has the result of the Nth Fibonacci number. The value of each register can be viewed by using the switches **sw[7:0]** and Leds **Led[7:0]** in the FPGA board. Clock **clk** (100 MHz clock) is used as input. Register R1 and R2 are used as intermediate registers. Register R0 which always has a value of 0 is used in the branch instruction for comparison.

|  |  |  |
| --- | --- | --- |
| Instruction | Instruction Memory (16 bit) | Comments |
| ADDI R3, R0, #5 | {ADDI, 3'h0, 3'h3, 7'd11} | R3 = 11 (N = 11) |
| ADDI R3, R3, #1 | {ADDI, 3'h3, 3'h3, 7'h1} | R3 = R3 + 1 |
| ADDI R1, R0, #1 | {ADDI, 3'h0, 3'h1, 7'h1} | R1 = 1 |
| SUBI R2, R0, #0 | {SUBI, 3'h0, 3'h2, 7'h1} | R2 = -1 |
| LOOP: ADD R1, R1, R2 | {ADD, 3'h1, 3'h2, 3'h1, 4'h0} | R1 = R1 + R2 (LOOP) |
| SUB R2, R1, R2 | {SUB, 3'h1, 3'h2, 3'h2, 4'h0} | R2 = R1 - R2 |
| SUBI R3, R3, #1 | {SUBI, 3'h3, 3'h3, 7'h1} | R3 = R3 - 1 |
| BNQ R3, R0, LOOP | {BNQ, 3'h3, 3'h0, 7'd3 << 1} | LOOP if R3 != R0 |
| STORE R1, 1(R0) | {STORE,3'h0, 3'h1, 7'h1} | [R0 + 1] = R1 |
| LOAD R4, 1(R0) | {LOAD, 3'h0, 3'h4, 7'h1} | R4 = [R0 + 1] = 0x37 |

# Simulation Results

N = 10. Register R4 and Led output has 0x37 (55 in decimal)

A screenshot of a computer

Description automatically generated

# Challenges

1. Installing Xilinx on Windows 10. Since Xilinx needs Virtual Machine to run with Windows, the installation process is rather complex and time consuming.
2. The ISIM simulator hangs frequently since it is running on Virtual Machine.
3. Sometimes, ISIM fails silently throwing a generic error. Searching for the error online is not very helpful because Xilinx ISE is no longer supported.
4. Xilinx Synthesis Tool throws many warnings, so I was challenging to see which are relevant.
5. Encountered Timing issues with FPGA. Eliminated all latches in the design and made Memory writes trigger during positive edge of the clock to fix the issue.

# Future Scope

1. More instructions can be implemented – CALL instruction, all Branch and Jump instructions and Logical and Arithmetic Operations.
2. Multi-cycle computer with Pipelining can be implemented. Control and Data Hazards needs to be handled in this case.