ECE 385

Spring 2021

Experiment #5

Simple Computer SLC-3 in SystemVerilog

Quanrui Bai, Jarod Partlo

ABJ

Xinbo Wu, Kevin Xia

Introduction

In this experiment, we used SystemVerilog and a DE10-Lite FPGA to design a simple 16-bit computer that functions similarly to an LC-3 machine. This microprocessor is capable of executing a subset of the instructions in the full LC-3 ISA.

The computer's CPU consists of a few primary components: a control unit, an ALU, a register unit of 16-bit registers, and a program counter unit. Within the control unit are several parts which are explained in more detail further in this report. In addition to the CPU, the microprocessor also contains provided memory and memory interface for storing data and instruction commands, and the input/output interface that reads inputs from the switches and prints data to the hex display.

Written Description and Diagrams of SLC-3

a. Summary of Operation

In each cycle of the computer, the processor fetches, decodes, and executes an instruction stored in the memory before beginning again. The first 4 bits of the instruction are passed to the Instruction Sequencer/Decoder Unit (ISDU) to identify which instruction to execute.

This computer uses a memory-mapped I/O model. Inputs (the board switches) and outputs (the 7-segment hex display) are mapped to memory address 0xFFFF. The Mem2IO module serves as the interface between the CPU and both the memory and the I/O devices.

Once the ISDU control unit has deciphered the instruction, it outputs the appropriate control signals to the other units of the computer. For instance, in State 18 (see Figure 4), which is part of the Fetch cycle, the ISDU will activate the MAR's load signal and

the GatePC, along with setting the PCMUX to read PC+1. Note, however, that the data bus gates are not true tristate buffers. To prevent the interference that tri-state buffers can cause and to simplify the design, the gates are implemented as a multiplexer. The multiplexer controls which signal is sent onto the bus. The gate signals are sent to the data bus unit which contains the mux, and the mux select signal is determined from which gate is set high.

Instruction	Instruction(15 downto 0)							Operation
ADD	0001	DR		SR1	0	00	SR2	R(DR) ← R(SR1) + R(SR2)
ADDi	0001	DR		SR	1	imm5		R(DR) ← R(SR) + SEXT(imm5)
AND	0101	DR		SR1	0	00	SR2	R(DR) ← R(SR1) AND R(SR2)
ANDi	0101	DR		SR	1	imm5		R(DR) ← R(SR) AND SEXT(imm5)
NOT	1001	DR		SR	111111			R(DR) ← NOT R(SR)
BR	0000	n	z	p	PCoffset9			if ((nzp AND NZP) != 0) PC ← PC + SEXT(PCoffset9)
JMP	1100	000		BaseR	000000			PC ← R(BaseR)
JSR	0100	1	PCoffset11					R(7) ← PC; PC ← PC + SEXT(PCoffset11)
LDR	0110	DR		BaseR	offset6			R(DR) ← M[R(BaseR) + SEXT(offset6)]
STR	0111	SR		BaseR	offset6			M[R(BaseR) + SEXT(offset6)] ← R(SR)
PAUSE	1101	ledVect12						LEDs ← ledVect12; Wait on Continue

Table -1 SLC-3 ISA table

b. Description of SLC-3 and Execution Cycle

For our simple computer, we will read instruction from onboard memory. After reset, the processor will stay at the HALTED state, and then we will press Run to enter the execution Cycle, Fetch-Decode-Execute.

Fetch

Firstly, we will load the PC value into MAR (Memory Address Register) by the control unit and data bus, so the MAR is the address pointed by the PC register. Then in

the next state, we will read the data from the address specified by the MAR, storing it in the MDR (Memory Data Register). This process is also processed by the control unit to set gate and load signal. In the final state of fetch, the value in the MDR will be sent into the IR (instruction register) by data bus to get the instruction for decoding.

Decode

We implement the decoder by combinational logic. The control unit will enable LD_BEN to get BEN signal from NZP register. The decoder will read the opcode from instruction to determine which operation the process does.

Execution

After decoding, the processor will do the specific operation by the ISDU state diagram. Each operation will have different execution state flow, but at last it will go back to halted state.

Instructions Performed(opcode)

ADD (0001): This instruction is to get the sum of two values after addition. Adds the contents of SR1 and SR2, and stores the result to DR. Sets the status register.

ADDi (0001): Add Immediate. Adds the contents of SR to the sign-extended value imm5, and stores the result to DR. Sets the status register.

AND (0101): This instruction is to get the bitwise AND result of two values. ANDs the contents of SR1 with SR2, and stores the result to DR. Sets the status register.

ANDi (0101): And Immediate. ANDs the contents of SR with the sign-extended value imm5, and stores the result to DR. Sets the status register.

NOT (1001): Negates SR and stores the result to DR. Sets the status register.

BR (0000): Branch. If any of the condition codes match the condition stored in the status register, takes the branch; otherwise, continues execution. (An unconditional jump can be specified by setting NZP to 111.) Branch location is determined by adding the sign-extended PCoffset9 to the PC.

JMP (1100): Jump. Copies memory address from BaseR to PC.

JSR (0100): Jump to Subroutine. Stores current PC to R(7), adds sign-extended PCoffset11 to PC.

LDR (0100): Load using Register offset addressing. Loads DR with memory contents pointed to by (BaseR + SEXT(offset6)). Sets the status register

STR (0111): Store using Register offset addressing. Stores the contents of SR at the memory location pointed to by (BaseR + SEXT(offset6)).

PAUSE (1101): Pauses execution until Continue is asserted by the user. Execution should only unpause if Continue is asserted during the current pause instruction; that is, when multiple pause instructions are encountered, only one should be “cleared” per press of Continue. While paused, ledVect12 is displayed on the board LEDs. See I/O Specification section for usage notes.

c. Block Diagram of slc3.sv

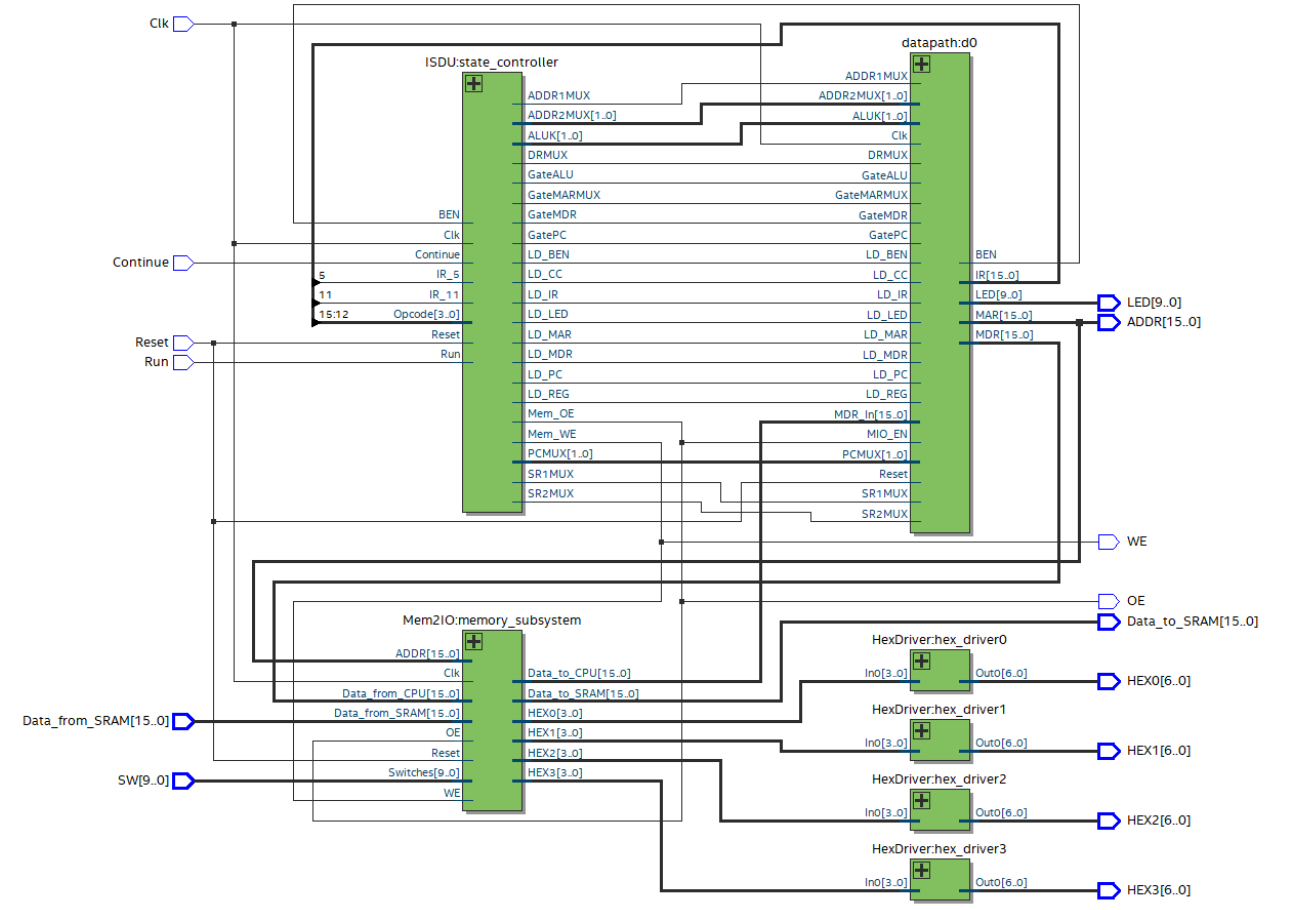

Figure 1 – Block Diagram of slc3.sv

d. Written description of all .sv modules

Module:slc3.sv
Inputs: [9:0] SW, Clk, Reset, Run, Continue, [15:0] Data_from_SRAM
Outputs: [9:0] LED,[6:0] HEX0, HEX1, HEX2, HEX3,[15:0] ADDR, [15:0] Data_to_SRAM, OE, WE
Description: This is the top-level design entity for our simple computer.
Purpose: This will wire up our modules of ISDU, Mem2IO, and datapath, and connect to LED display, switches, and two keys. And the block diagram of slc3.sv is on the above.

Module:reg_16.sv
Inputs: [15:0] Data_In, Clk, Reset, Load
Outputs: [15:0] Data_Out
Description: This module is for shift register, with positive edge triggered 16-bit register.
Purpose: This module is to accept the shift enable, shift in, and reset logic to control the register whether to shift, load, or reset.

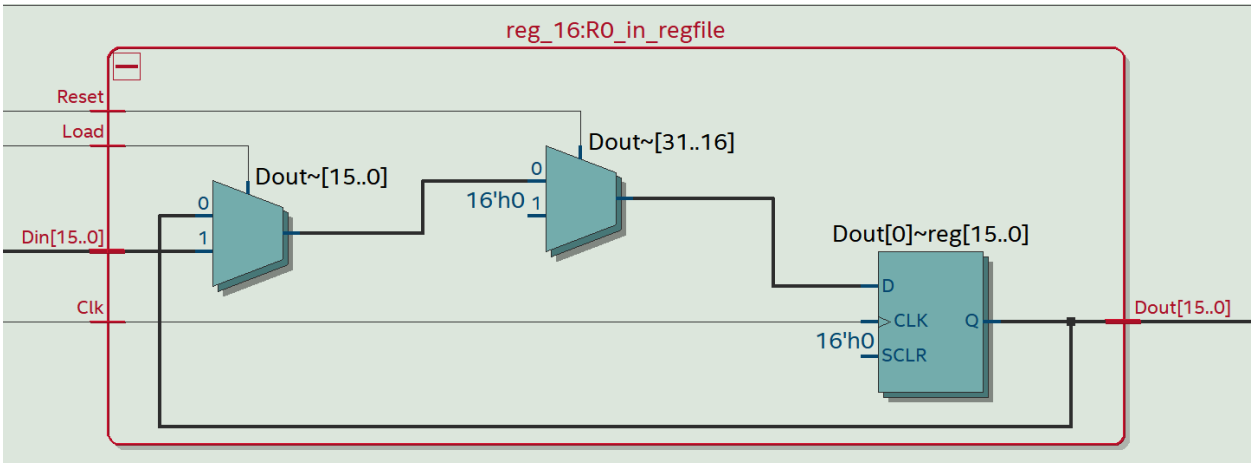

Figure -2 Block Diagram of reg_16.sv

Module:register_file.sv
Inputs: Clk, Reset, LD_REG, [15:0] BUS_val, [2:0] DR_val, SR1_val, SR2_val,
Outputs: [15:0] SR1_Out, SR2_Out
Description: This file is for the 8 registers, which is depend on the input value for Destination Register and Source Register.
Purpose: This will transfer the value from data bus to register based on the load signal from ISDU states.

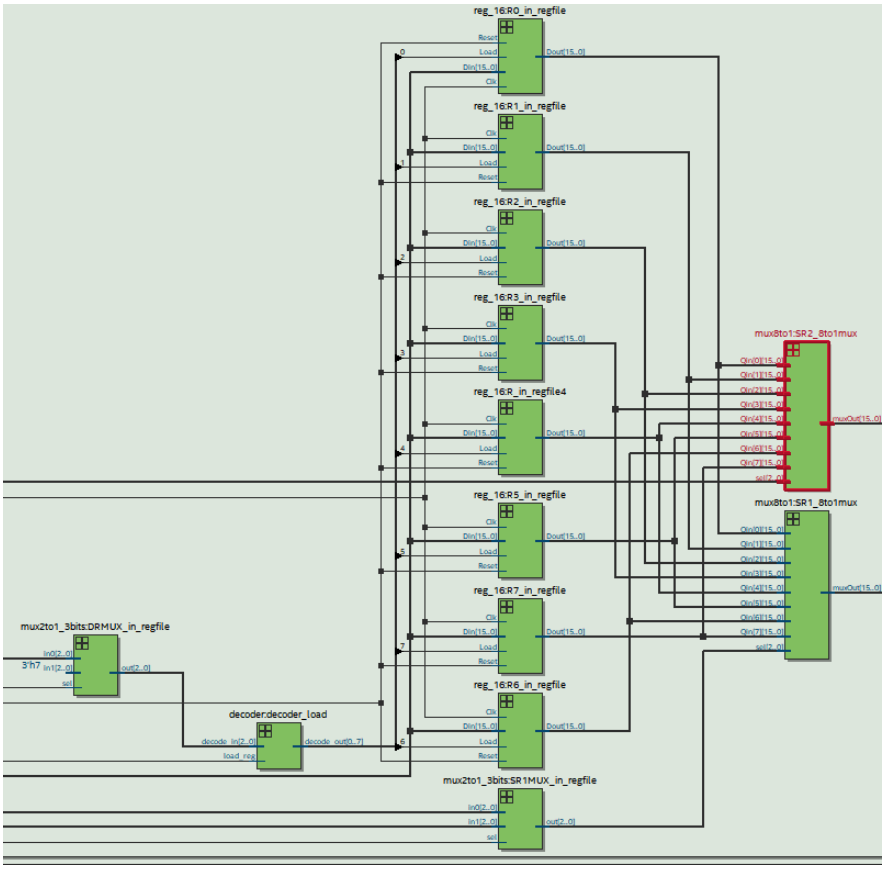

Figure -3 Block Diagram of register_file.sv

Module:mux_2.sv
Inputs: S, logic [n-1:0] In_0, In_1
Outputs: [n-1:0]Out
Description: This file is for the 2 to 1 Multiplexer for n bits. It uses dynamic parameter for more bits.
Purpose: This module will select one outputs from two inputs based on the select signal.

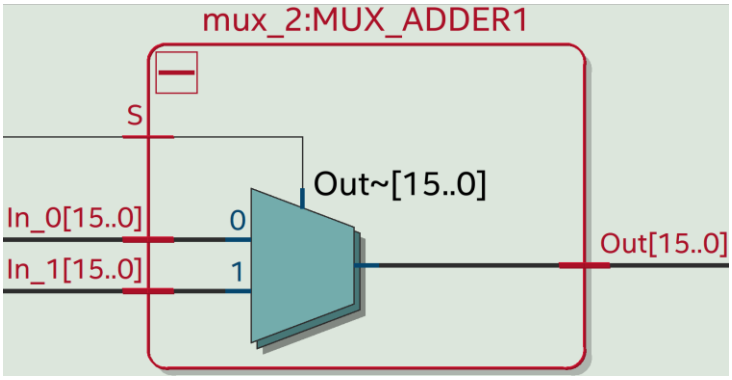

Figure -4 Block Diagram of mux_2.sv

Module:mux_4.sv
Inputs: S, logic [n-1:0] In_0, In_1,In_2, In_3, In_4
Outputs: [n-1:0]Out
Description: This file is for the 4 to 1 Multiplexer for n bits. It uses dynamic parameter for more bits.
Purpose: This module will select one outputs from four inputs based on the select signal.

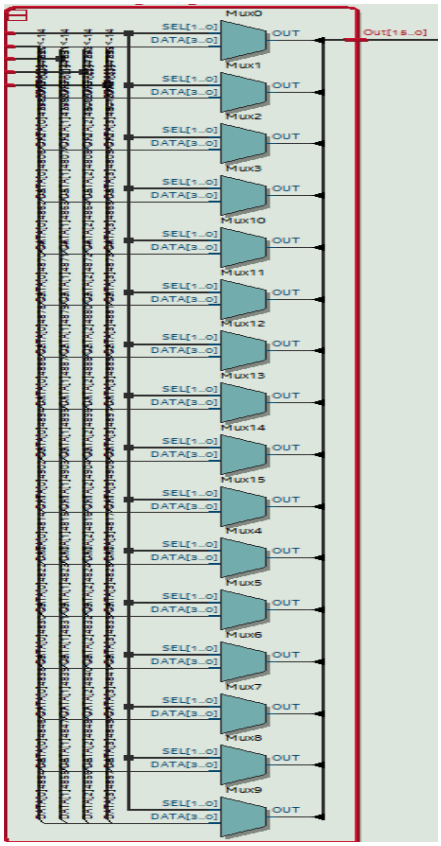

Figure -5 Block Diagram of mux_4.sv

Module:mux_bus.sv
Inputs: [3:0]S, logic [15:0] In_0, In_1,In_2, In_3, In_4
Outputs: [15:0]Out
Description: This file is for the 4 to 1 Multiplexer for four gates. (GateMDR, GateALU, GateMARMUX, and GatePC)
Purpose: It will control which gate is open based on the four bit select signal.

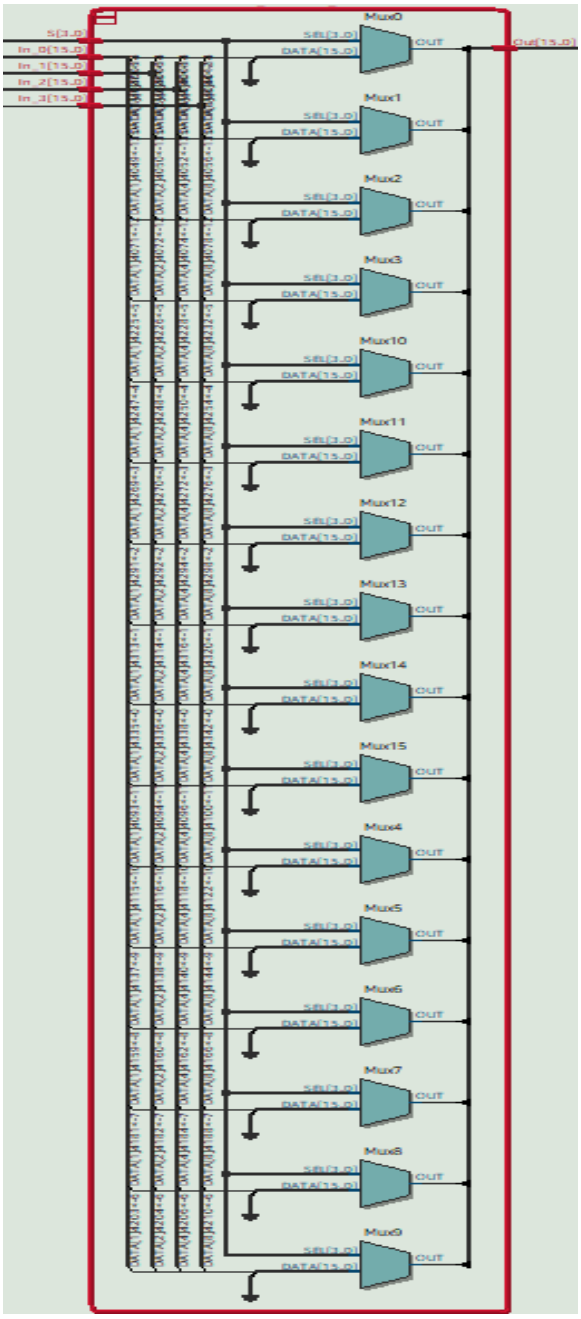

Figure -6 Block Diagram of mux_bus.sv

Module: HexDriver.sv
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: This module has hardcoded values that convert 4bit binary numbers from registers into hexadecimal
Purpose: It will support the display hexadecimal onto FPGA board.

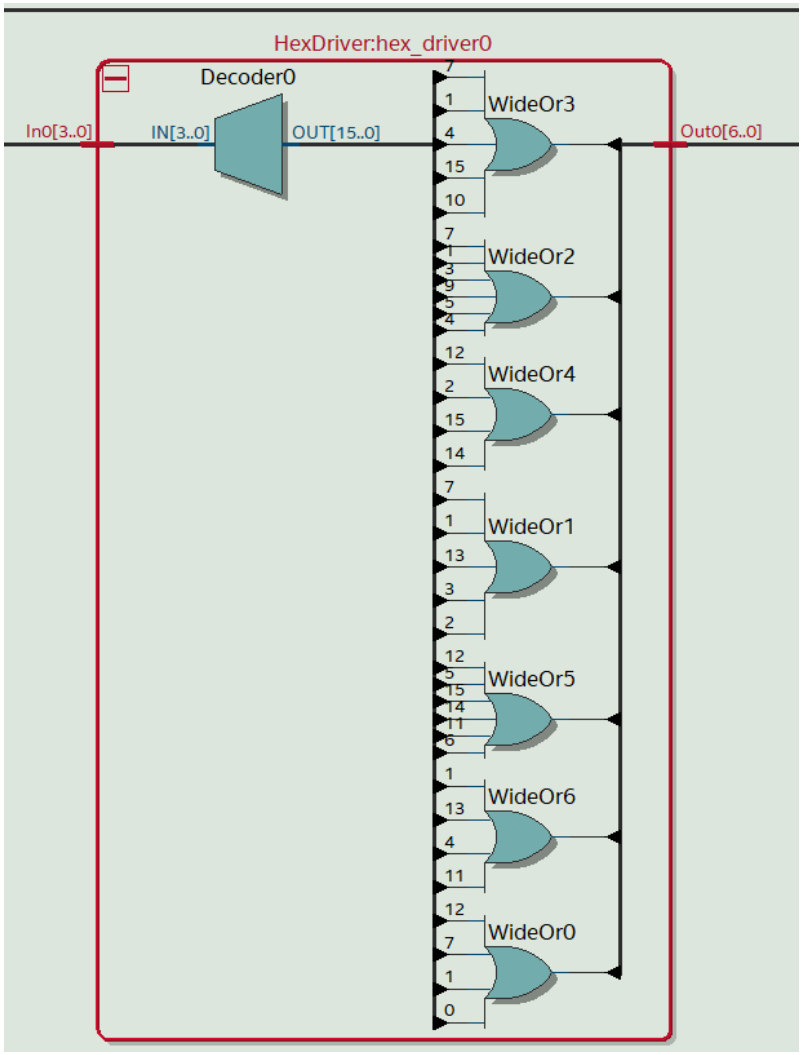

Figure -7 Block Diagram of HexDriver.sv

Module: test_memory.sv
Inputs: Reset,Clk, [15:0] data, [9:0] address, rden, wren
Outputs: [15:0] readout
Description: This module is for simulation.
Purpose: In simulation, this memory is guaranteed to work at least as well as the actual memory. (no block diagram)

Module: Mem2IO.sv
Inputs: Clk, Reset,[15:0] ADDR, OE, WE,[9:0] Switches, [15:0] Data_from_CPU, Data_from_SRAM
Outputs: [15:0] Data_to_CPU, Data_to_SRAM,[3:0] HEX0, HEX1, HEX2, HEX3
Description: This module is the interface for our DE10-Lite FPGA board IO and the SRAM.
Purpose: It will take the input from switches and send the output to Hex display. Both are on the 0xFFFF physical address.

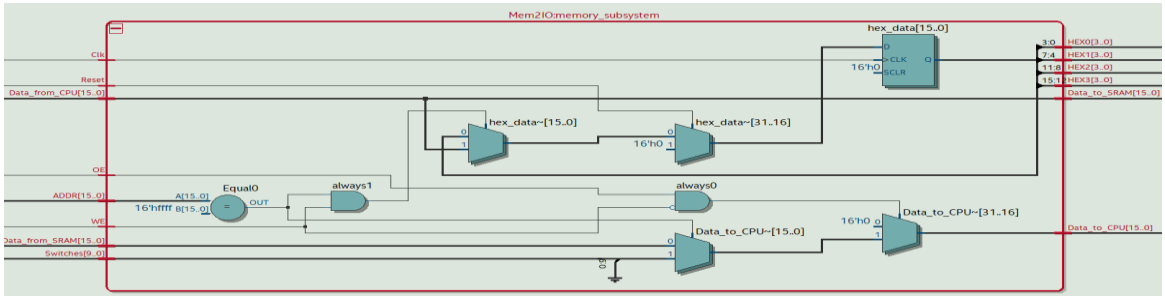

Figure -8 Block Diagram of Mem2IO.sv

Module: ISDU.sv
Inputs: : Clk, Reset, Run, Continue, [3:0] Opcode, IR_5, IR_11, BEN
Outputs: : LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, PC_LED, GatePC, GateMDR, GateMAR,GateALU [1:0] PCMUX, ADDR2MUX, ALUK, SR1MUX, SR2MUX, ADDR1MUX, Mem_OE, MEM_WE
Description: This module is the state machine for our simple computer. It will set load, gate, signal for our datapath.
Purpose: After decoding, the ISDU will enter each path based on different instructions.

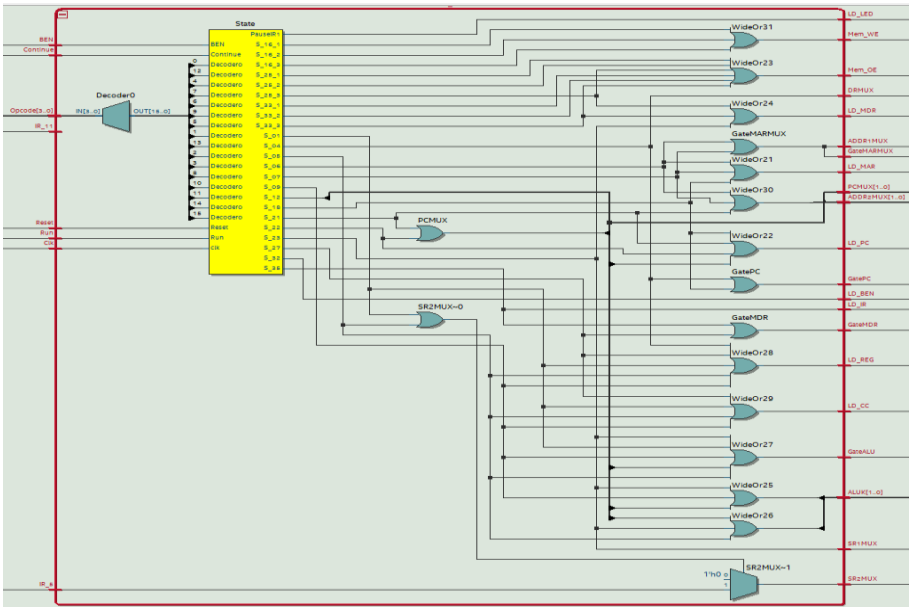

Figure -9 Block Diagram of ISDU.sv

Description of the operation of the ISDU

The ISDU (Instruction Sequencer/Decoder Unit) takes in the instruction in segments as several logic inputs. It then uses a series of cases and conditionals to set outputs – load signals, gate signals, mux inputs, etc. - appropriately for the instruction. An always_ff block sets State to Next_state each clock cycle unless Reset is pressed, in which case State goes to Halted. State and Next_state are enumerated logic which represent each of the state bubbles in Figure 4 under the State Diagram of ISDU section of this report.

An always_comb block contains the unique case (State) statement of the ISDU, whose central component is the OpCode decoder (another switch-case statement) under case State 32 (S_32). State 32 is reached after the instruction has been fetched and is where it is decoded.

This OpCode decoder takes the OpCode input as 4-bit logic binary, taken from the 4 most significant bits read from the memory address, and directs the ISDU to the correct execution state. Each OpCode input corresponds to the instruction listed in Table 1. For example, if the OpCode is 4'b0001, this represents an ADD instruction, where the sum of two source registers is placed in a destination register. Each case in this decoder sets the Next_state to the proper execution state, and on the next clock cycle, State gets this Next_state (when Reset has not been pressed).

When the unique case (State) statement is assessed again, mux select outputs of the ISDU are set according to the actions described in the execution state bubble in Figure 4. When communication needs to occur over the databus, the register load signals and gate signals are set accordingly. At the beginning of the block, all outputs are zeroed, so the only lines inside the case are those setting the right outputs high.

Module: datapath.sv
Inputs: Clk, Reset, LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, GateMDR, GatePC, GateALU,GateMARMUX, MIO_EN,SR2MUX, ADDR1MUX, MARMUX, DRMUX, SR1MUX, [1:0] PCMUX, ADDR2MUX, ALUK, [15:0] MDR_In
Outputs: [15:0] IR_Out, PC_Out, MAR_Out, MDR_Out, BEN, [9:0] LED
Description: This module is the integrated module for multiplexer, ALU, BEN and data bus.
Purpose: It will connect every module in the SLC-3 for our simple computer.

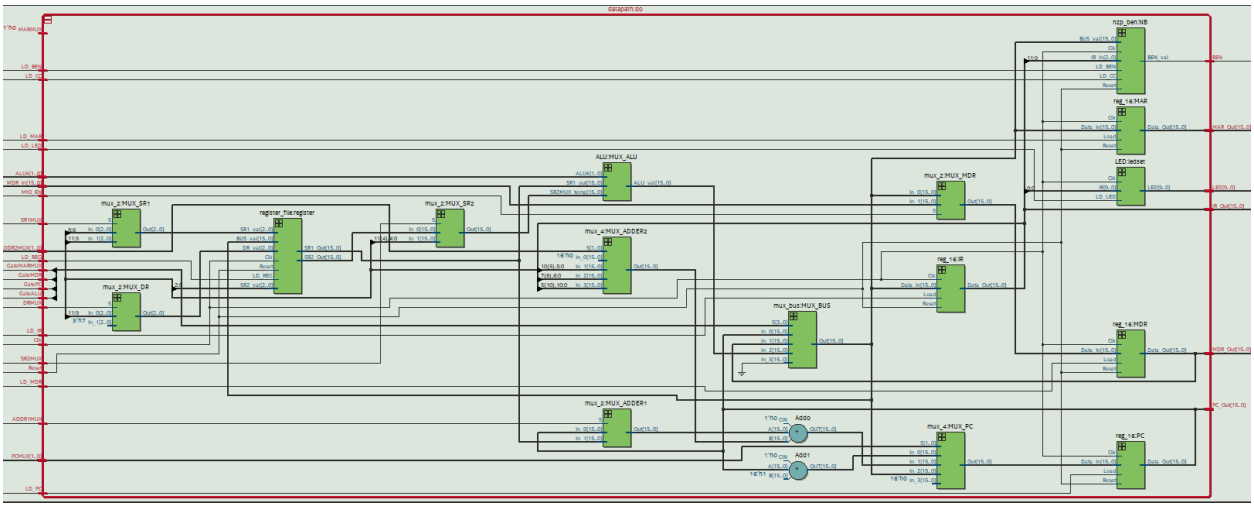

Figure -10 Block Diagram of datapath.sv

Module: Synchronizers.sv
Inputs: Clk, (Reset), d
Outputs: q
Description: This module is provided synchronizers file.
Purpose: The synchronizers are used to prevent meta-stabilities for certain critical signals due to nonideal flip-flop behaviors.

Module: SLC3_2.sv
Inputs: NONE
Outputs: NONE
Description: This is the library for all SLC-3 instructions divided by all functions based on the opcode.
Purpose: This library will be used for memory_contents for simulation and debugging.

Module: ALU.sv
Inputs: [15:0] SR1_out,SR2MUX_temp, [1:0]ALUK,
Outputs: [15:0] ALU_val
Description: This module is for four operations from two 16-bit inputs.
Purpose: It will perform ADD, AND, NOT, and PASS operations based on the select signal.

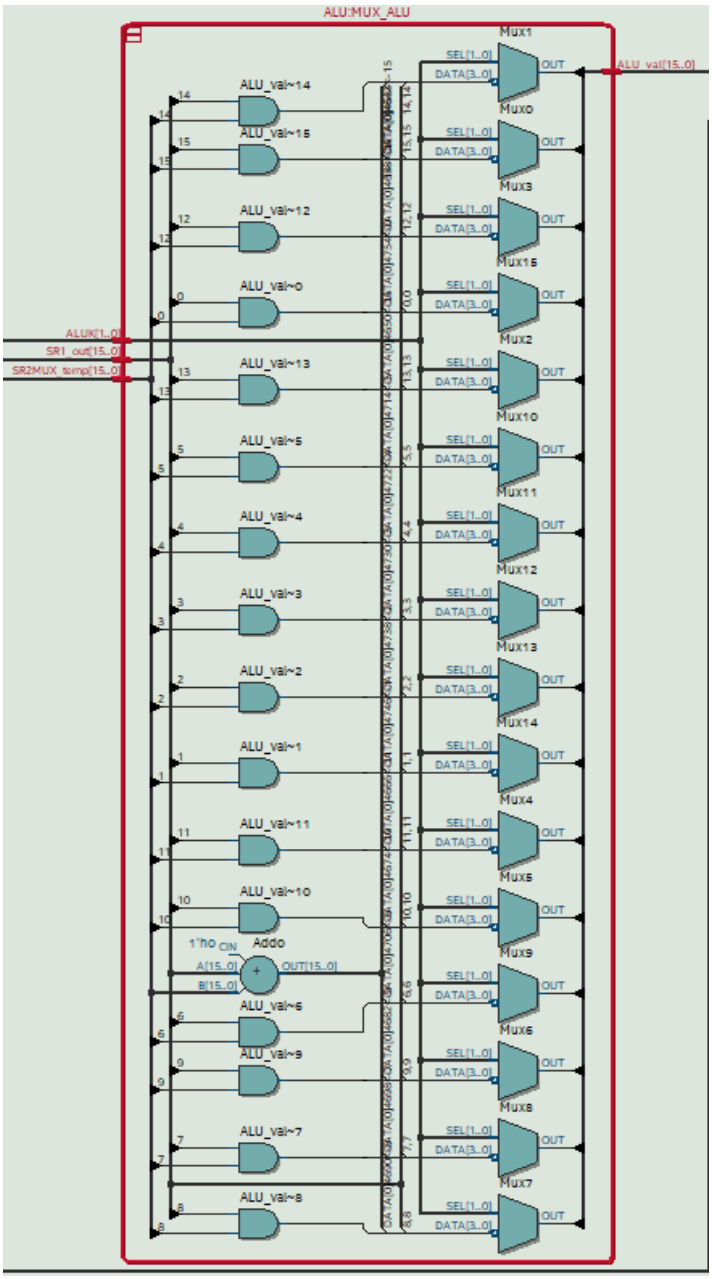

Figure -11 Block Diagram of ALU.sv

Module:nzp_ben.sv
Inputs: Clk, Reset, LD_CC, LD_BEN, [15:0] BUS_val, [2:0] IR_in,
Outputs: BEN_val
Description: This module is for checking the status of register after each instruction.
Purpose: It will check the data from the data bus is zero, positive, or negative, and then output the BEN value when the LD_BEN is high.

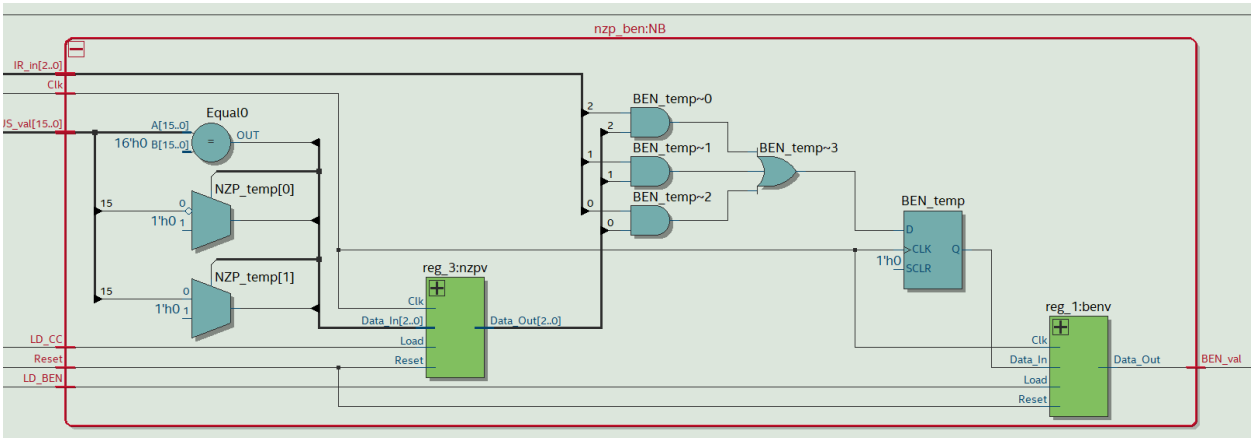

Figure -12 Block Diagram of nzp_ben.sv

Module:LED.sv
Inputs: Clk, LD_LED, [9:0] IR
Outputs: [9:0] LED
Description: This module is to show the last 10 bits of IR.
Purpose: This module will connect the LED with IR last 10 bits.

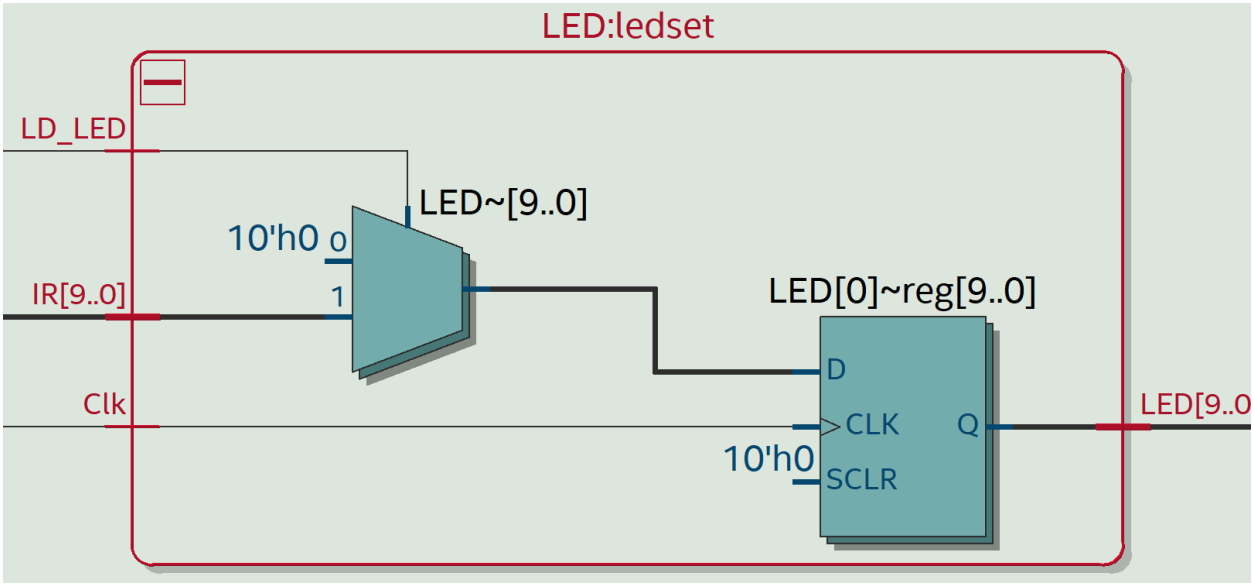

Figure -13 Block Diagram of LED.sv

e. State Diagram of ISDU

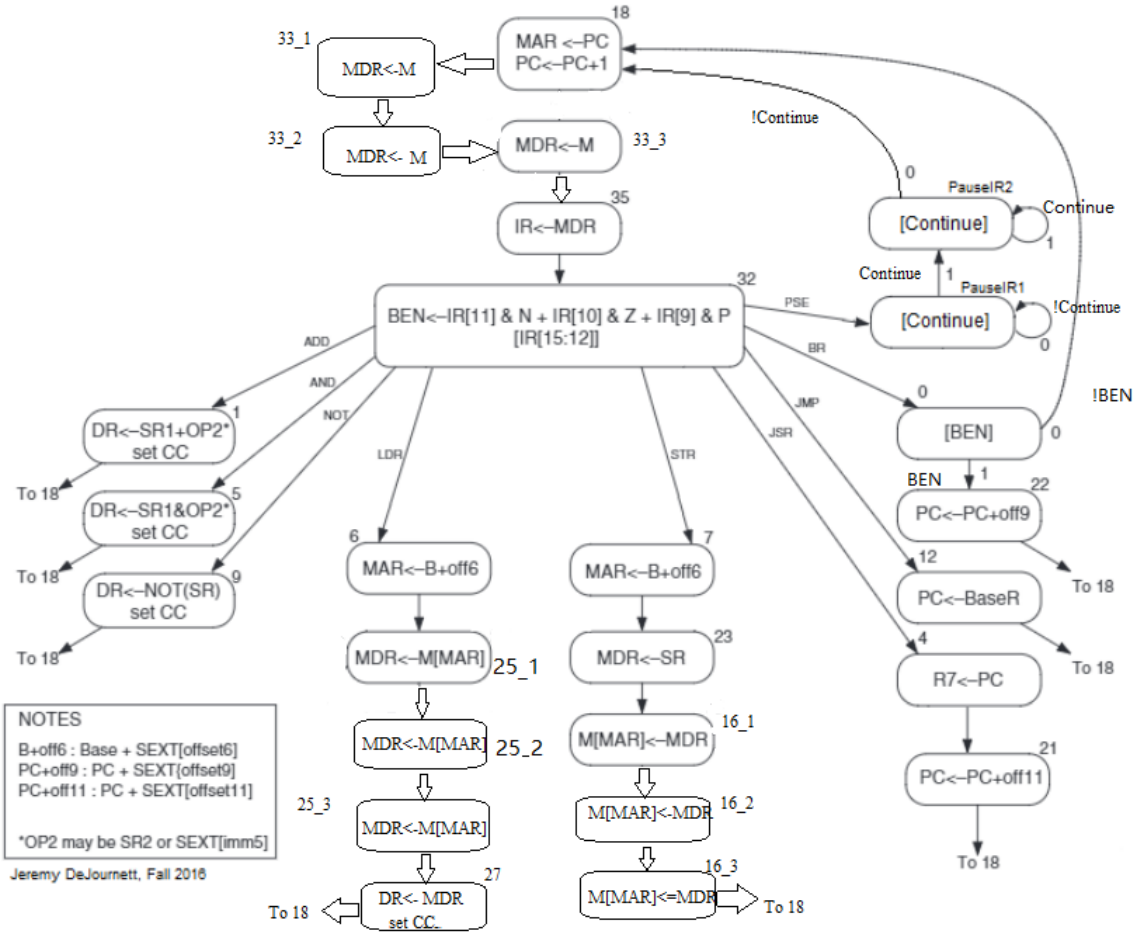

Figure 14- State Diagram of ISDU

Simulations of SLC-3 Instructions

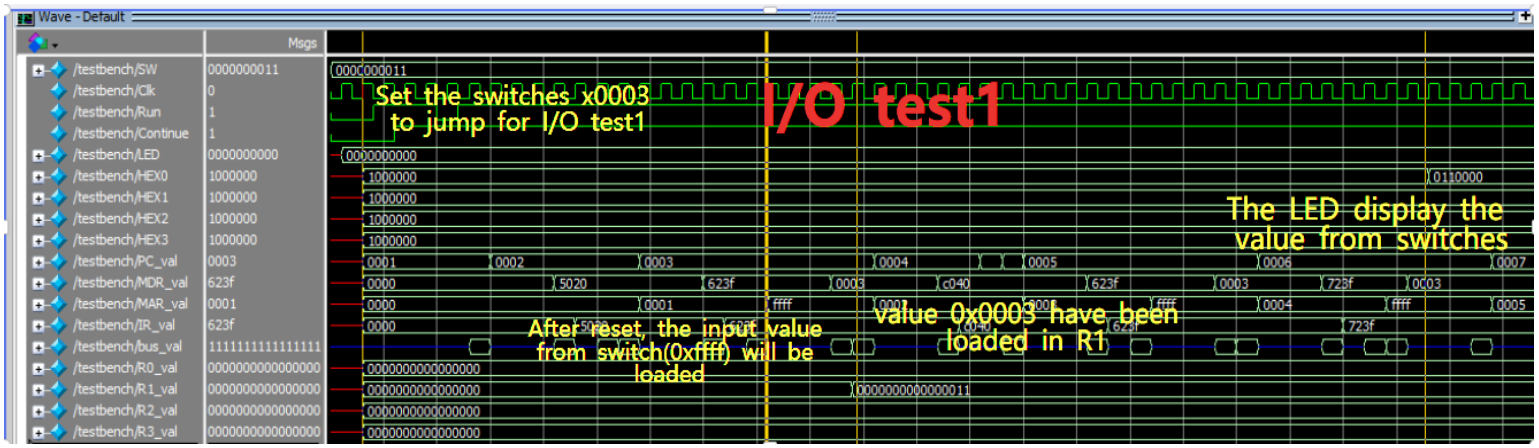

Figure -15 Simulation of I/O test1

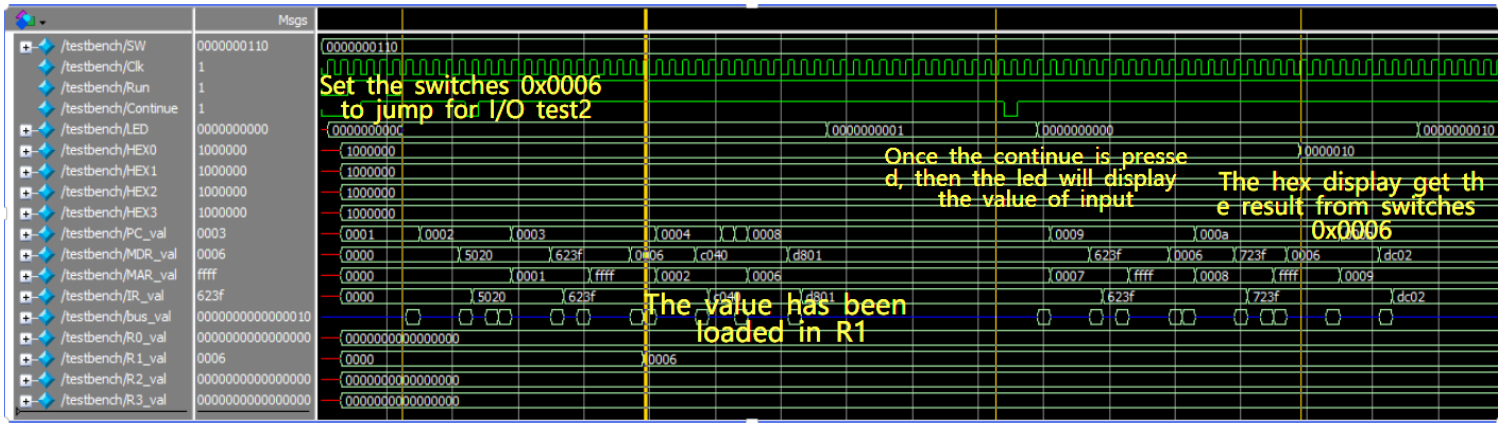

Figure -16 Simulation of I/O test2

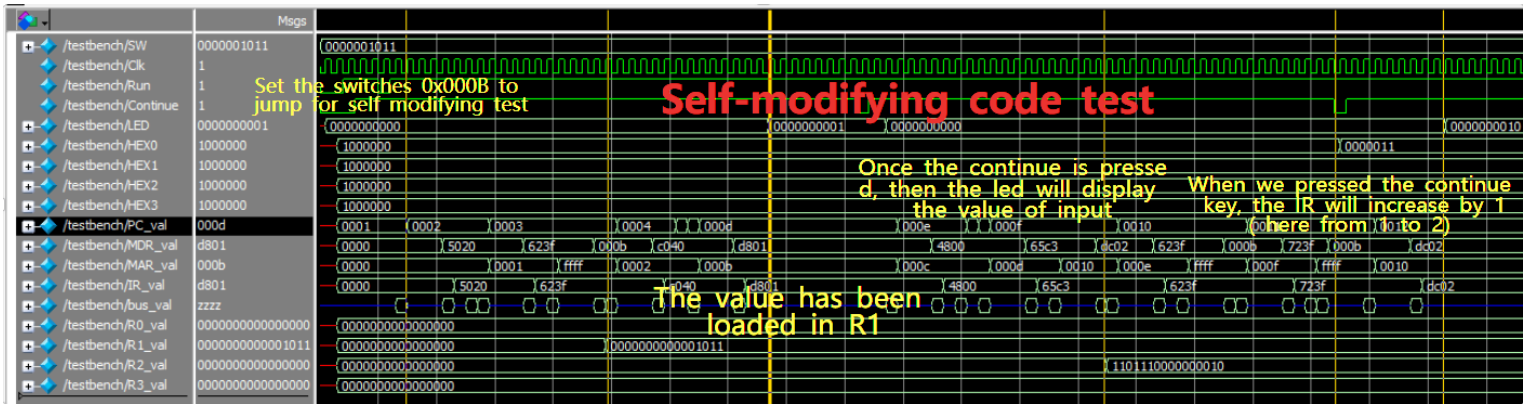

Figure -17 Simulation of Self-modifying code test

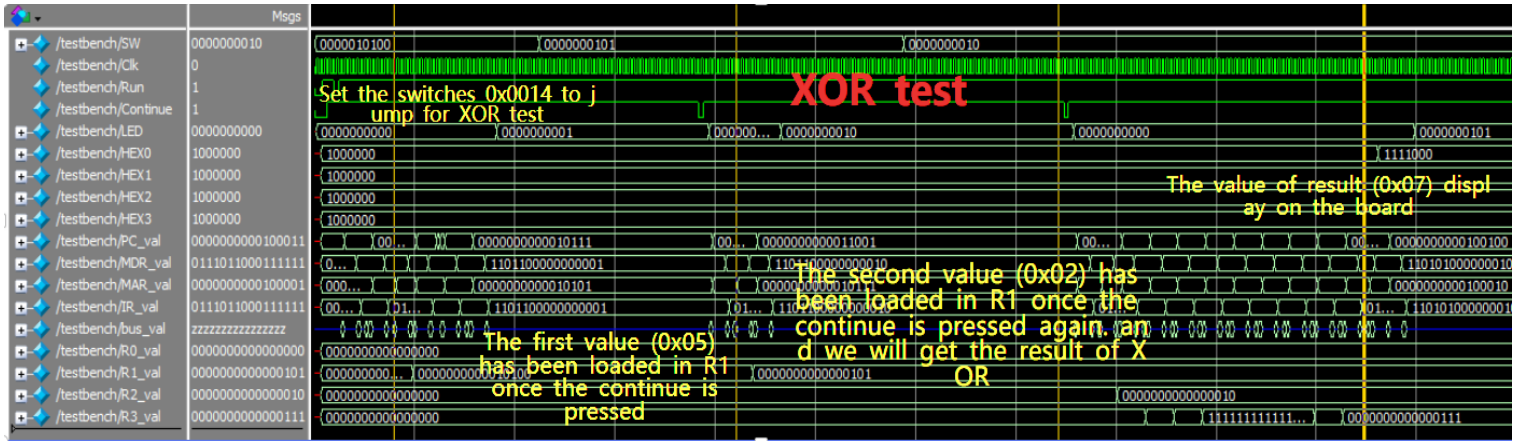

Figure -18 Simulation of XOR test

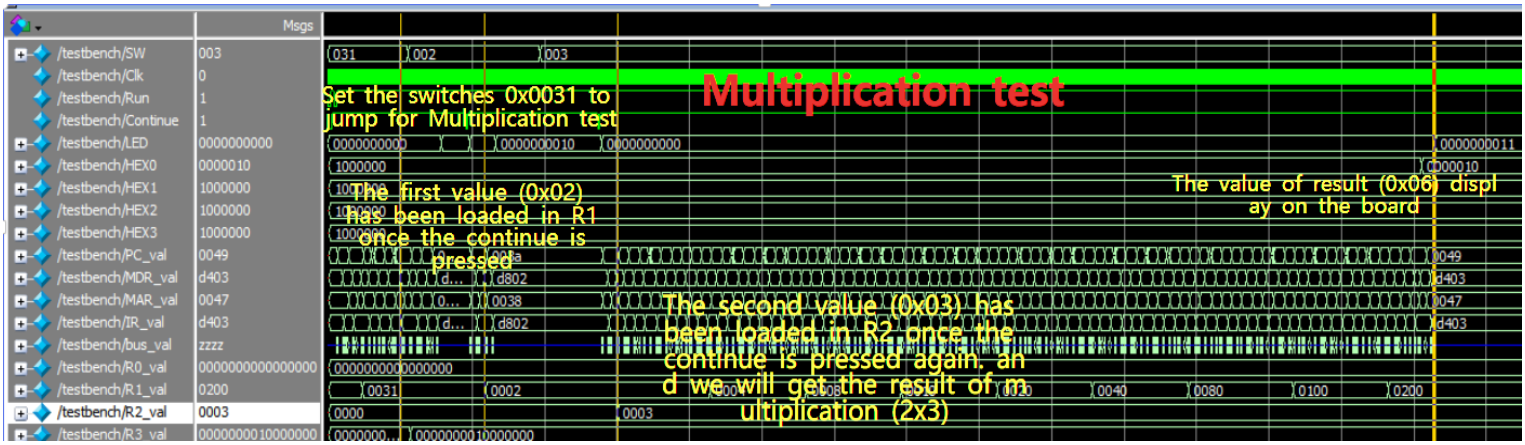

Figure -19 Simulation of Multiplication test

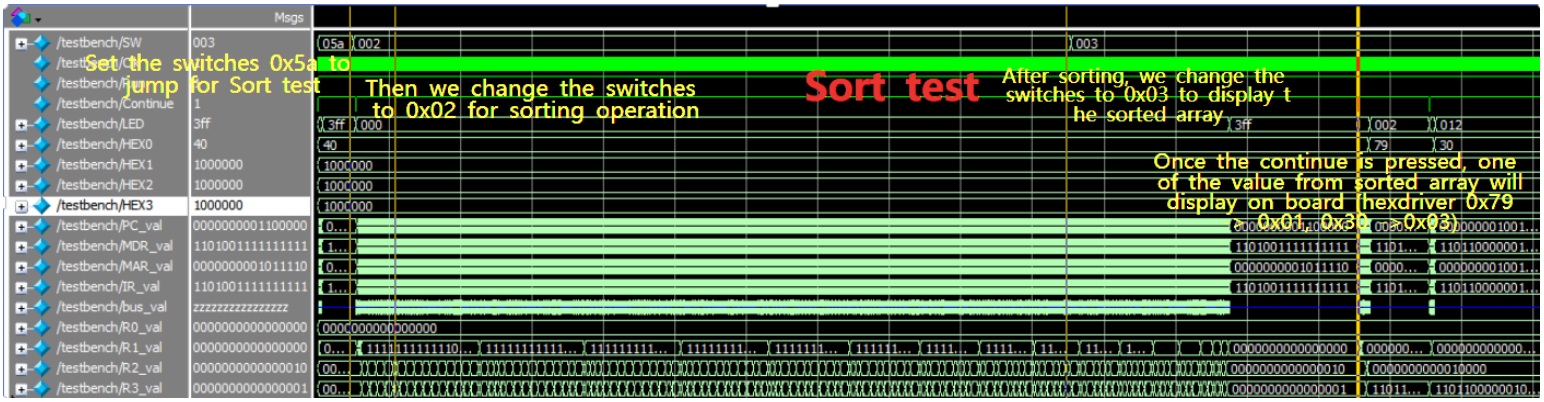

Figure -20 Simulation of Sort test

Post-Lab Questions

a. Design Resources and Statistics Table

LUT	871
DSP	0
Memory (BRAM)	0
Flip-Flop	269
Frequency (Mhz)	92.41
Static Power (mW)	89.94
Dynamic Power (mW)	0.00
Total Power (mW)	98.63

Table -2 Design Resources and Statistics Table

a. Questions

(1) Mem2IO is used to allow the CPU to send and receive from the input/output. In this implementation, we use memory-mapped I/O, so both the input from the switches and the output to the hex display are mapped to the same memory address. To get input from the switches, we perform a load from address 0xFFFF, and to write to the display, we store to address 0xFFFF. Without it, the CPU could access that location in memory, but wouldn't be able to receive inputs or outputs. It allows both read and write of the memory module and effectively intercepts and interprets calls to the I/O address.

(2) The JMP instruction sets the PC to some other memory address and stores the current PC in Register 7. The BR instruction is a conditionally jump instruction based on the NZP flag. BR does not store the current PC value. And BR only takes 9 bits of pc offset, but JMP takes 11bits of pc offset.

(3) The R signal (Memory Ready) in Patt and Patel is a signal from the memory module informing the CPU whether it is ready for another read or write. Since all components in our design have the same clock signal, rather than have an R signal, we use up to 3 states (doing same thing with W/R signal) to ensure the memory data is ready to be read or written, which means we will wait for two extra clock cycles until the memory has its work done. The implication of this design is that there is a upper limit frequency to how fast the computer can run. "Overclocking" would break the design.

Conclusion

In this lab, we designed and built a 16-bit computer on the DE10-Lite FPGA for executing a simplified version of the LC-3 ISA. Our computer consists of a CPU, a memory unit and memory and input/output interface.

Within the CPU, there are several modules, including an ALU, a collection of 16-bit registers, and a control unit (ISDU). The control unit is a state machine that is sent the relevant portions of the instruction read from memory and decodes them, setting the next state accordingly and outputting register load signals, gate control signals, mux select signals to execute the encoded command.

The memory interface allows the CPU to read and write memory, but also allows it to read from and write to input/output peripherals using memory-mapped I/O. Memory address 0xFFFF is reserved for I/O, with a read sending data to the CPU from the onboard switches and a write taking data from the CPU and outputting it on the hex display.

We don’t recommend changes for this lab. The lab was very challenging but a valuable experience.