ECE 385

Fall 2023

Experiment 5

Designing a LC-3 using Systemverilog

Cher Rui Tan, Adithya Balaji
Section: Friday 11.30am
TA: Gene Lee

Introduction

In this lab we build a SLC-3 processor, which is a simple microprocessor. It is a subset of the LC-3 ISA, which is a 16-bit processor. It contains a 16-bit Program Counter (PC), which is a register that keeps track of the next instruction to be executed. Additionally, it contains a 16-bit Instruction register (IR) that acts as codes to tell the microprocessor what instruction to execute by accessing the relevant registers, memory locations and any additional signals. Finally, it contains multiple other 16-bit registers, which are elaborated further within the lab report.

Written Description and Diagrams of SLC-3

The SLC-3 always starts in State 18 where it performs a Fetch operation, where it pulls the instruction at the PC value from memory. This is then followed by the Decode state where it determines using the Op-code where it should branch to to execute what operation. After one the 11 unique operations have been performed, the State machine returns to the Fetch state to carry out the next operation. This is repeated till all instructions have been completed.

The SLC-3 is a (RISC) Reduced Instruction Set Computer. Which can execute the following functions: register and immediate ADDs, register and immediate ANDs, NOT, Branch, Jump, JSR, Loads, Stores and has the additional Pause state for user readability.

Block Diagram of slc3.sv

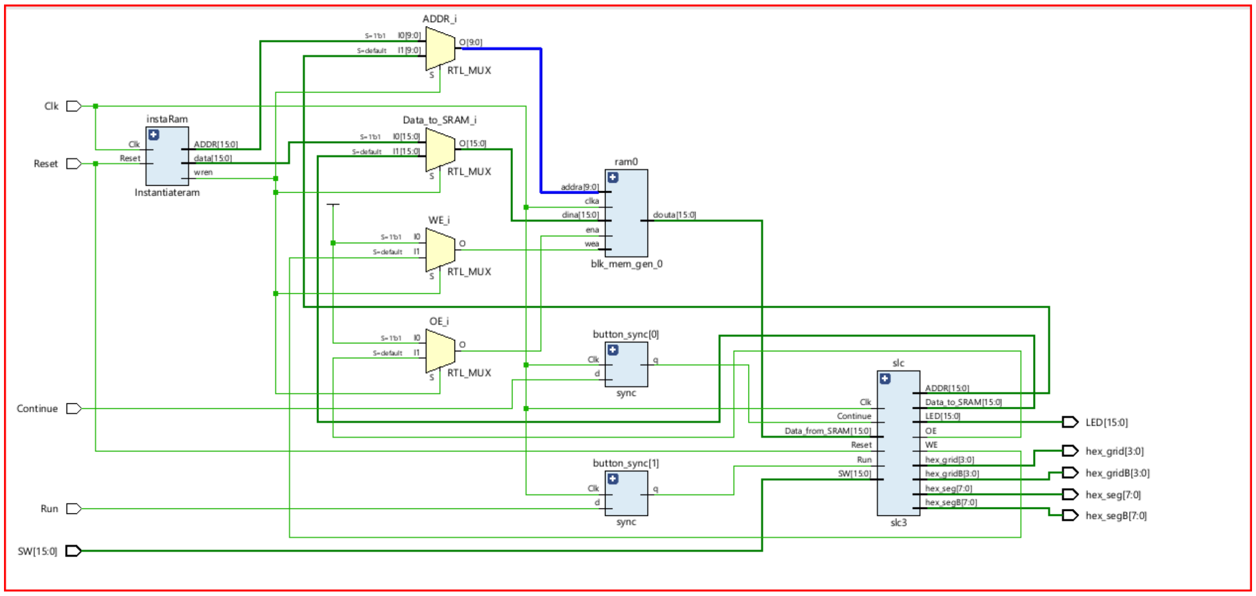

Figure 1: RTL block diagram of LC3

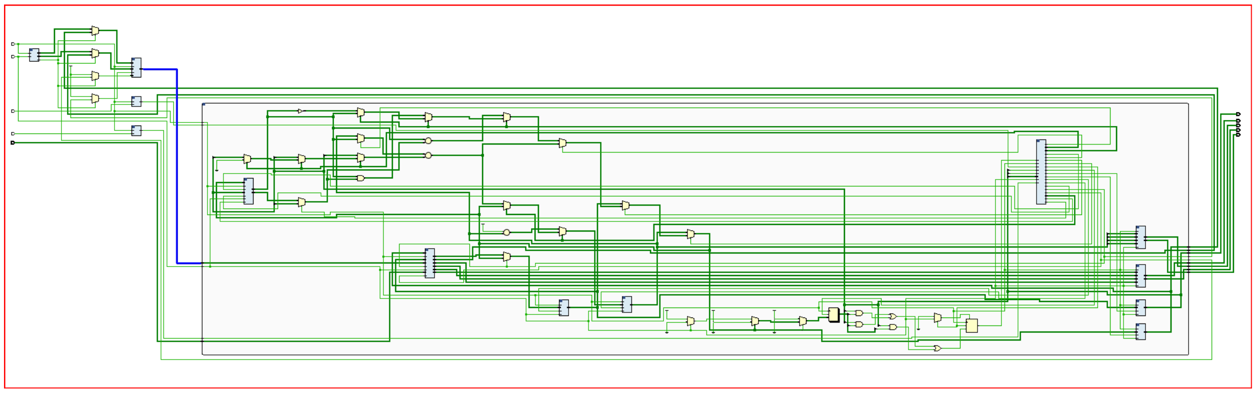

Figure 2: Block diagram for slc3.sv

Written Description of all .sv modules

Module: slc3_sramtop.sv:

Inputs:	input logic [15:0] SW,
	input logic Clk, Reset,Run, Continue,
Outputs:	output logic [15:0] LED,
	output logic [7:0] hex_seg,
	output logic [3:0] hex_grid,
	output logic [7:0] hex_segB,
	output logic [3:0] hex_gridB

Description: instantiates the RAM, instantiates physical on chip memory, and contains always-comb logic for reading from and/or writing to memory

Purpose: Top level module to instantiate all the necessary components for the rest of the design

Module: slc3.sv:

Inputs:	input logic [15:0] SW,
	input logic Clk, Reset, Run, Continue,
	input logic [15:0] Data_from_SRAM,
Outputs:	output logic [15:0] LED,
	output logic OE, WE,
	output logic [7:0] hex_seg,
	output logic [3:0] hex_grid,

output logic [7:0] hex_segB,
output logic [3:0] hex_gridB,
output logic [15:0] ADDR,
output logic [15:0] Data_to_SRAM

Description: Instantiates Mem2IO, ISDU statecontroller, and handles logic for every MUX that exists in the LC3. Also handles the logic for setting NZP based on the last used register. Also assigns LED

Purpose: Acts as a primary level for the entire LC3, particularly in handling combinational logic.

Module: hex.sv:

Inputs:	input logic clk,
	input logic reset,
	input logic [3:0] in[4],
Output:	output logic [7:0] hex_seg,
	output logic [3:0] hex_grid

Description: Implements a 4-digit hexadecimal display driver by using a counter to cycle through 4 digits, and selecting the appropriate pattern based on the input nibbles. Also contains a module within for nibble to hex.

Purpose: To display hexadecimal numbers on the 4-digit 7-segment display

Module: nibble_to_hex (.sv)

Inputs:	input logic [3:0] nibble,
Outputs:	output logic [7:0] hex

Description: Takes asynchronous signals and holds them in a flip flop allowing for them to be implemented on a clock signal. This results in the signals being converted from asynchronous to synchronous signals.

Purpose: To eliminate asynchronous components such that the entire circuit remains synchronous to reduce the chance of static hazards arising from timing issues and ensure proper signal handling in FPGA designs.

Module: nibble_to_hex (.sv)

Inputs: input logic [3:0] nibble,
Outputs: output logic [7:0] hex

Description: Takes asynchronous signals and holds them in a flip flop allowing for them to be implemented on a clock signal. This results in the signals being converted from asynchronous to synchronous signals.

Purpose: To eliminate asynchronous components such that the entire circuit remains synchronous to reduce the chance of static hazards arising from timing issues and ensure proper signal handling in FPGA designs.

<u>Module: reg_17 (.sv)</u>

Inputs: input Clk, Reset, Load,
input [15:0] D,
Outputs: output logic [15:0] Data_Out

Description: creates a 16 bit register on the positive edge of the clock to be used/instantiated for all the relevant registers in the lc3, which are the PC, MAR, MDR and IR.

Purpose: Standard 16-bit register module to be used in instantiation

<u>Module: regfile (.sv)</u>

Inputs: input logic Clk,
input logic SR1MUX, DRMUX,
input logic LD_REG,
input logic [15:0] BUS,
input logic [15:0] IR,
input logic Reset,
Outputs: output logic [15:0] SR1,
output logic [15:0] SR2

Description: Stores data of R0 to R7. Contains DRMUX and SRMUX as registers which chooses different registers to store/load data based on the IR.

Purpose: Instantiate registers and the values they hold

<u>Module: MEM2IO (.sv)</u>

Input: input logic Clk, Reset,
input logic [15:0] ADDR,
input logic OE, WE,
input logic [15:0] Switches,
input logic [15:0] Data_from_CPU, Data_from_SRAM,
Outputs: output logic [15:0] Data_to_CPU, Data_to_SRAM,
output logic [3:0] HEX0, HEX1, HEX2, HEX3

Description: Memory mapped IO, that handles the logic for outputs and inputs by using read and write. Loading the memory address xFFFF loads the value of the switches into the CPU, else it loads the data from SRAM. Stores by writing to LED when address is xFFFF, else it resets. Additionally, assign the HEX.

Purpose: Memory mapped IO to handle input/output.

Module: ISDU (.sv)

Inputs: input logic Clk, Reset, Run, Continue
input logic[3:0] Opcode,
input logic IR_5,
input logic IR_11,
input logic BEN,

Outputs: output logic LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, _REG,
LD_PC, LD_LED,
output logic GatePC, GateMDR, GateALU, GateMARMUX,

output logic [1:0] PCMUX,
output logic DRMUX, SR1MUX, SR2MUX, ADDR1MUX,
output logic [1:0] ADDR2MUX, ALUK,
output logic Mem_OE, Mem_WE

Description: Implements the actual FSM and the handling of the next state logic. Additionally, sets MUX select signals within each state of the LC3 FSm to be able to perform the necessary operation.
Purpose: FSM logic design to implement the fetch, decode and execute in the LC3.

Module: Instantiateram (.sv)
Input: input Reset,

	input Clk,
Output:	output logic [15:0] ADDR,
	output logic wren,
	output logic [15:0] data;

Description: Creates a block memory ram on the lc3 board. When reset is pressed, it clears all memories and reloads the test programs into the block mem gen.
Purpose: To allocate memory block on the FPGA

Module: sync.sv:

Inputs:	input Clk, Reset_Clear, Run_Accumulate,
	input [7:0]SW,
Outputs:	output logic [7:0] Aval, Bval,
	output logic Xval,
	output logic [7:0] hex_segA,
	output logic [3:0] hex_gridA

Description: Takes asynchronous signals and holds them in a flip flop allowing for them to be implemented on a clock signal. This results in the signals being converted from asynchronous to synchronous signals.

Purpose: To eliminate asynchronous components such that the entire circuit remains synchronous to reduce the chance of static hazards arising from timing issues and ensure proper signal handling in FPGA designs.

Module: slc3testtop.sv:

Inputs:	input logic [15:0] SW,
	input logic Clk, Reset,Run, Continue,
Outputs:	output logic [15:0] LED,
	output logic [7:0] hex_seg,
	output logic [3:0] hex_grid,
	output logic [7:0] hex_segB,
	output logic [3:0] hex_gridB

Description: Acts to instantiate a “virtual” ram to perform simulation without actually using the hardware.

Purpose: To be used for simulation testing the FSM.

Module: testmemory.sv:

Inputs:	input Reset,
	input Clk,
	input [15:0] data,
	input [9:0] address,
	input ena,
	input wren,
Outputs:	output logic [15:0] readout

Description: Used as the memory contents so that the code can be run on the simulation without requiring any actual hardware.

Purpose: To be used for simulation testing the FSM.

Module: memorycontents.sv:

Description: Contains the opcodes and instructions that are used to test the LC3 in the simulation.

Purpose: To be used for simulation testing the FSM.

Module: mux_4_1.sv:

Inputs:	input logic [width-3:0] d0,
	input logic [1:0] s,
Outputs:	output logic [width-3:0] y;

Description: Creates a 4-bit to 1-bit multiplexer with a 2 bit select input. To be instantiated in the above modules.
Purpose: Standard 4-to-1 MUX.

Description of the operation of the ISDU (Instruction Sequence Decoder Unit)

The ISDU contains the following, fetch, decode and execute.

Fetch: State 18, State 33 (consisting of 3 holding states), state 35. This outputs the PC to the BUS, loads MAR with PC, and PC is incremented for the program to run, and instruction to be executed is pulled from memory and loaded into MDR, subsequently the MDR outputs this to the IR.

Decode: State 32. Load BEN is set to high in this state, and nzp is used to calculate BEN with the formula on the datapath. The nzp is determined every time a load is done. This nzp may or may not be used in the following instruction execution (depending on the IR). Opcode is used to decide which state to go to next, in the execute stage.

Execute: The ISDU is the FSM that contains every possible of the LC3 datapath instruction set. It contains the signal logics in each of the states such as gate, load, and mux signals, as well as write and read signals. Most of the state transitions are handled by the datapath. For example, for the STR, after the fetch and decode, it goes to state 7, then state 23, then state 16 (consisting of 3 holding states) before returning to state 18 where the next instruction is pulled from memory, and this is directly from the datapath. The ISDU only has the inputs of reset, run and continue, as well as BEN and IR.

State Diagram of ISDU

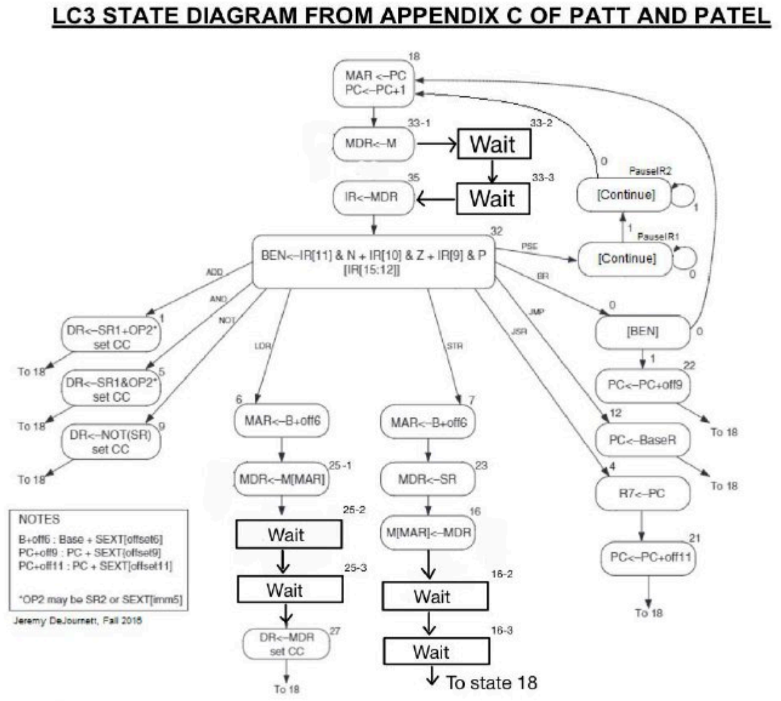

Figure 3: ISDU state diagram

Simulations of SLC-3 Instructions

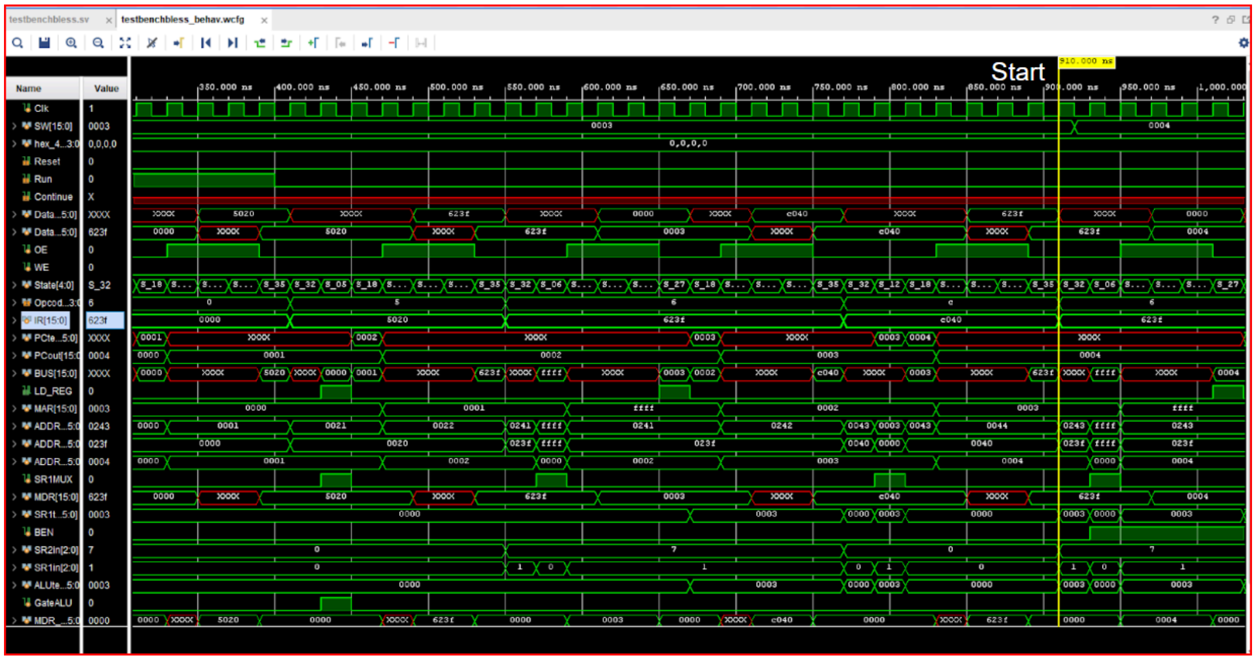

Figure 4: Initial before I/O Test 1

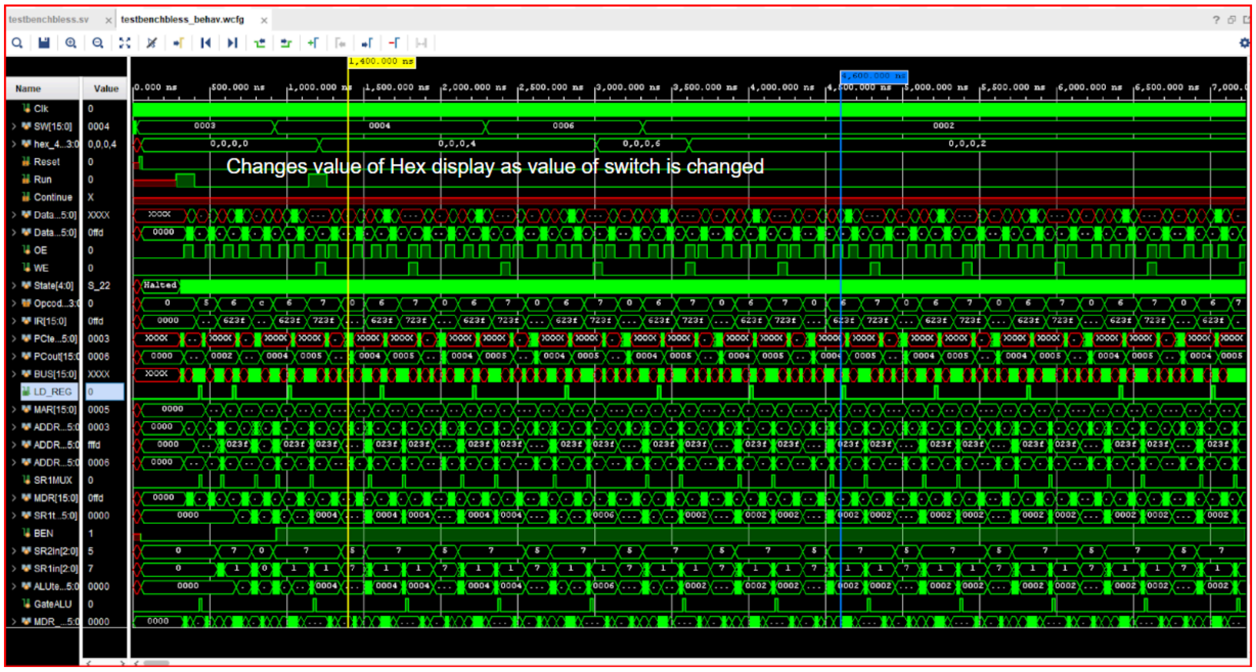

Figure 5: I/O Test 1 Display of the Switch Values being updated

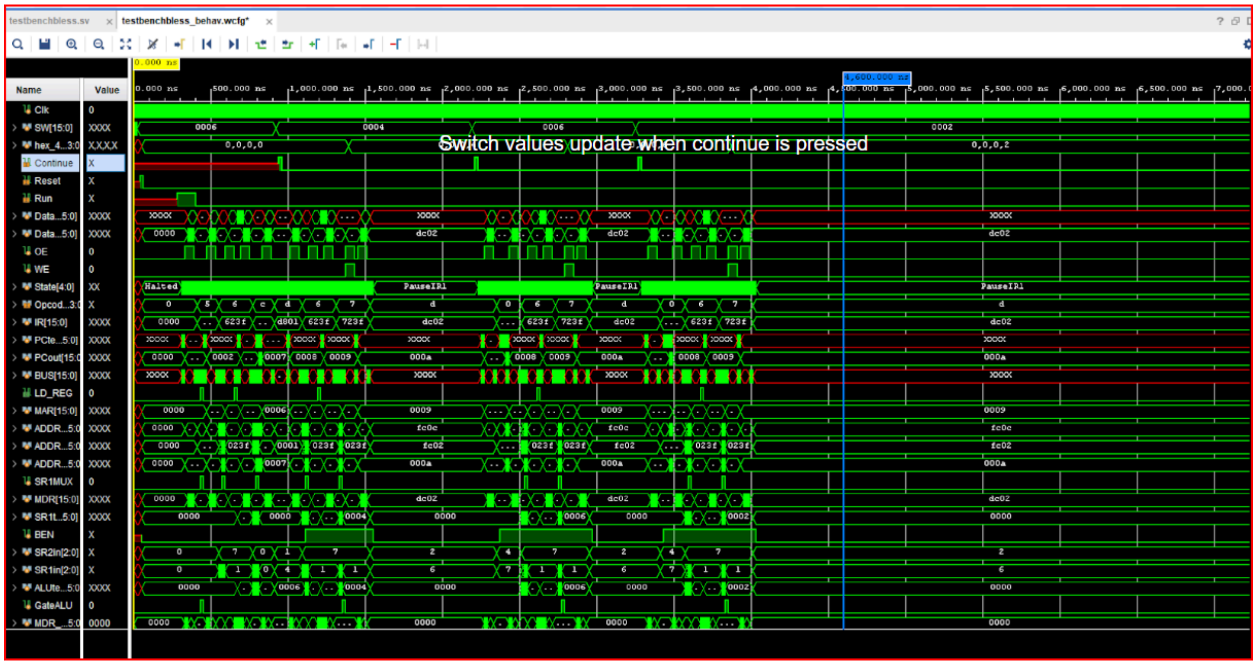

Figure 6: I/O Test 2 Switch values updated when continue is high

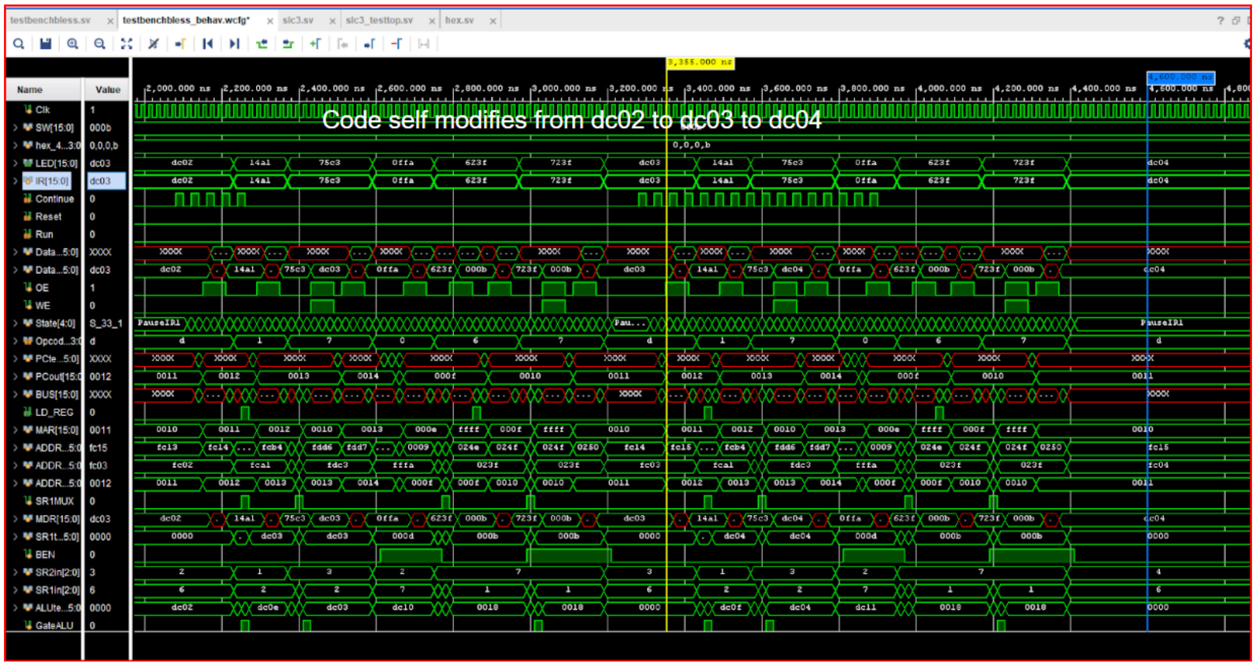

Figure 7: Self Modifying code where the dc02 becomes dc03 and then dc04

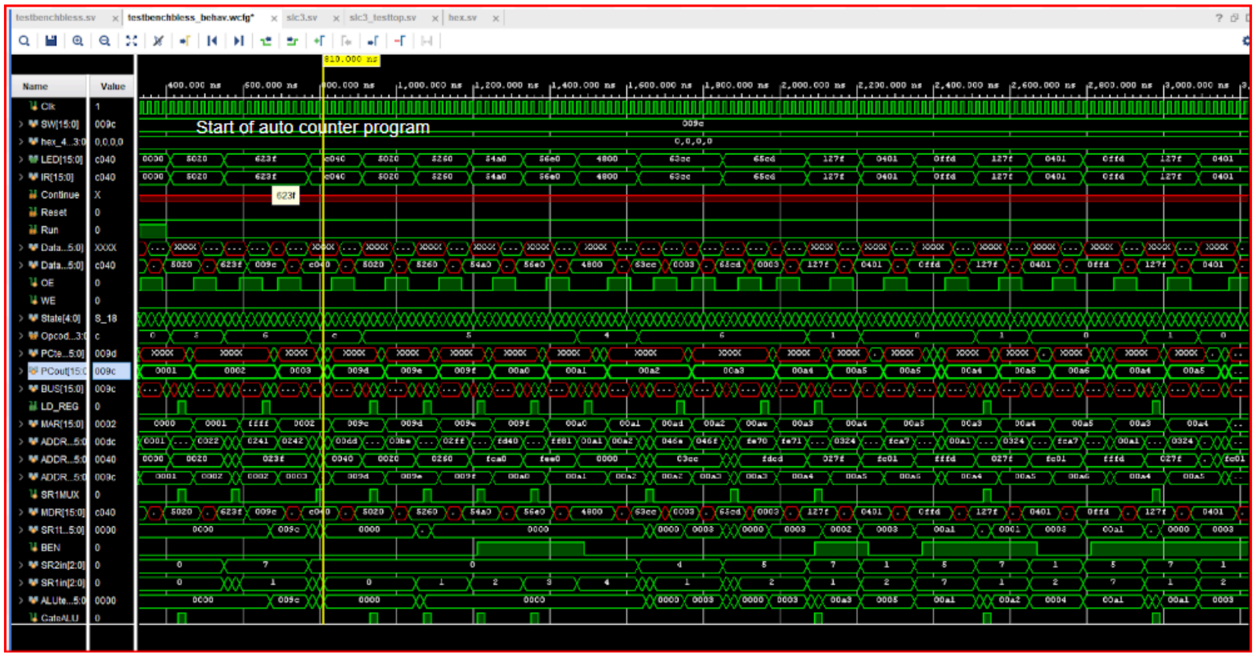

Figure 8: Start of Auto Counter

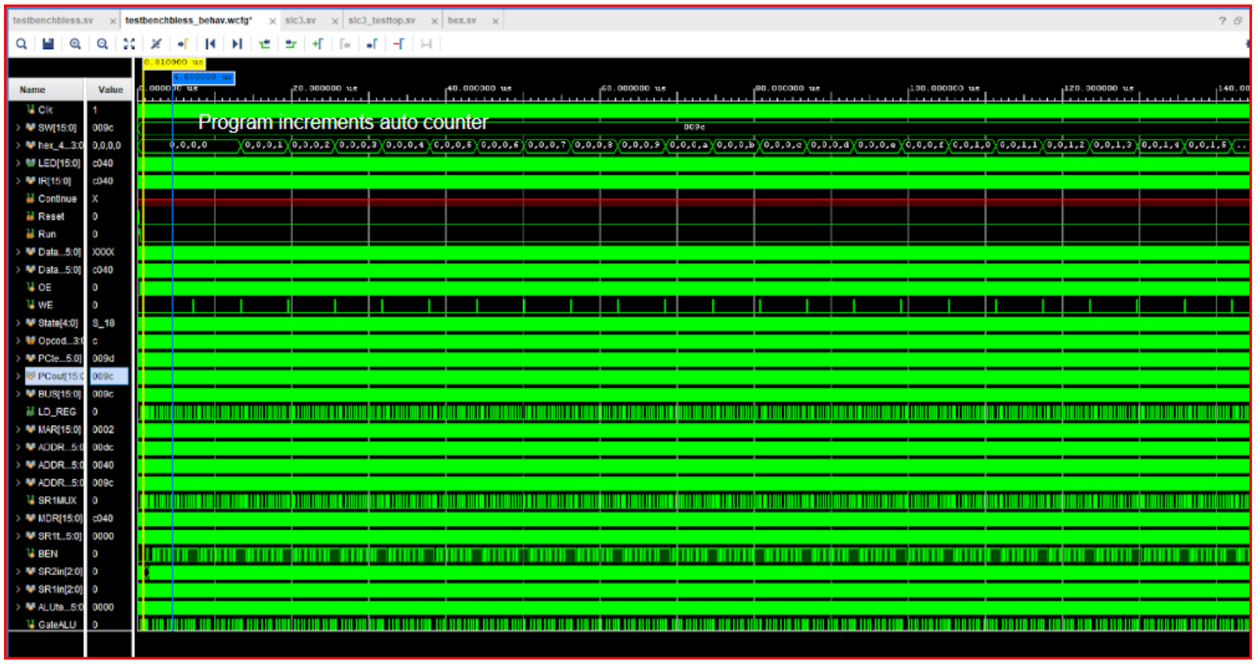

Figure 9: Display Auto Counter in Hex

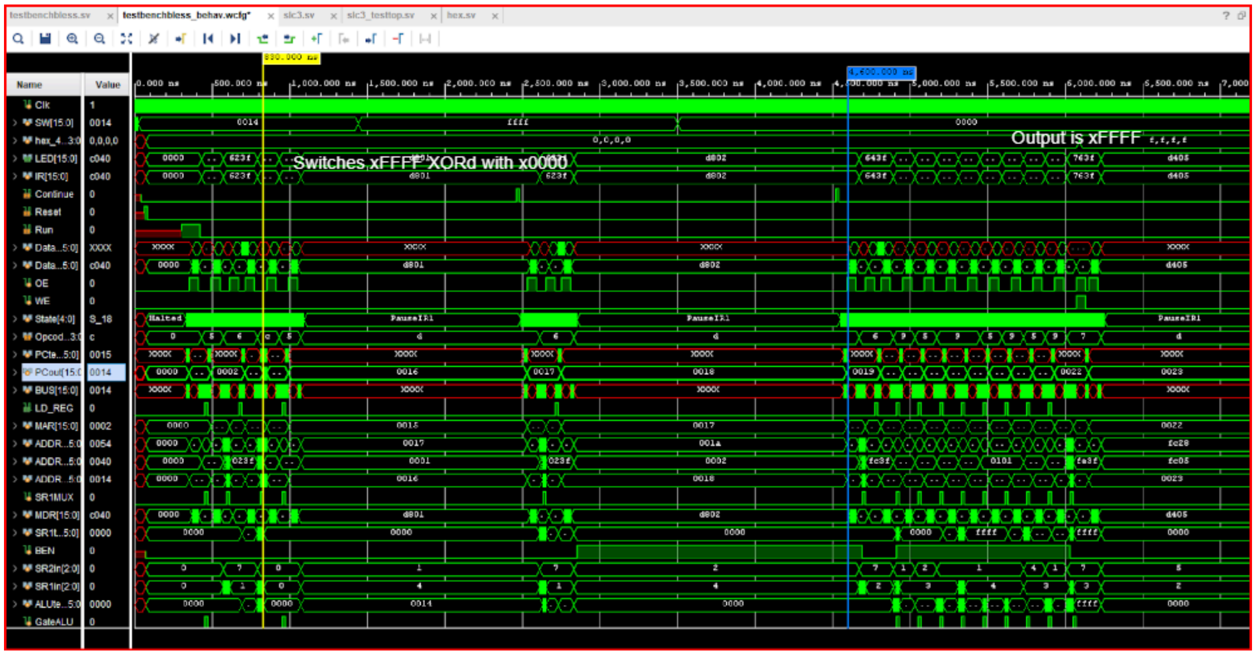

Figure 10: XOR test

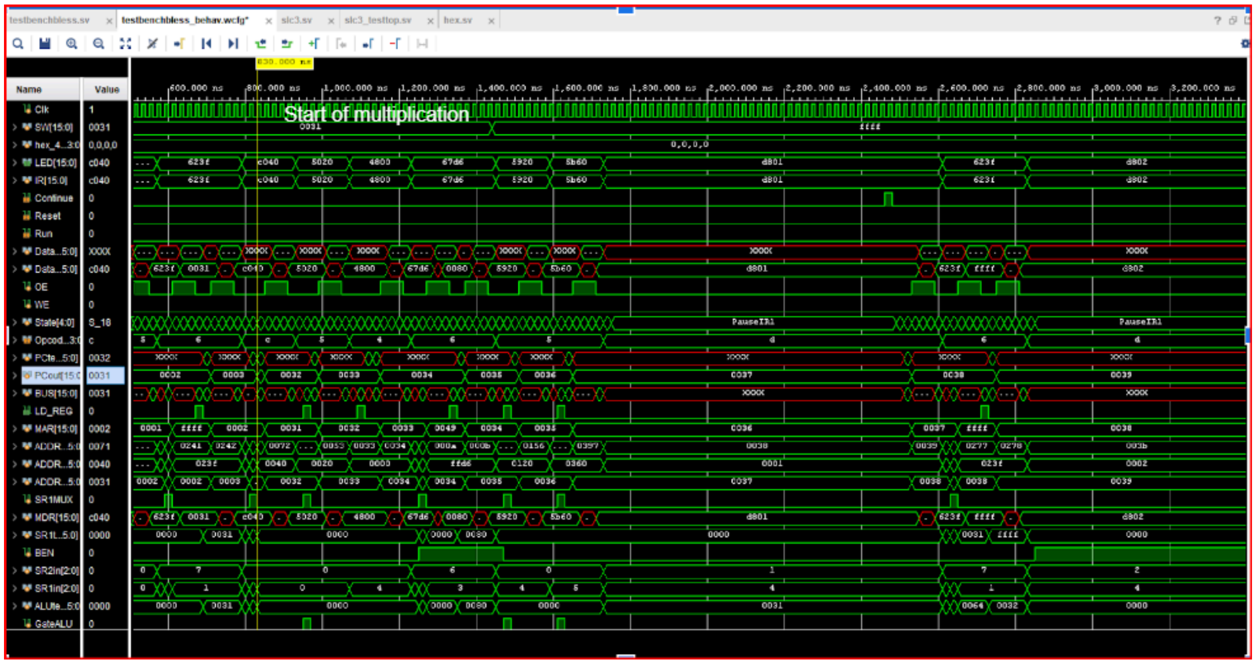

Figure 11 : Start of Multiplication Test

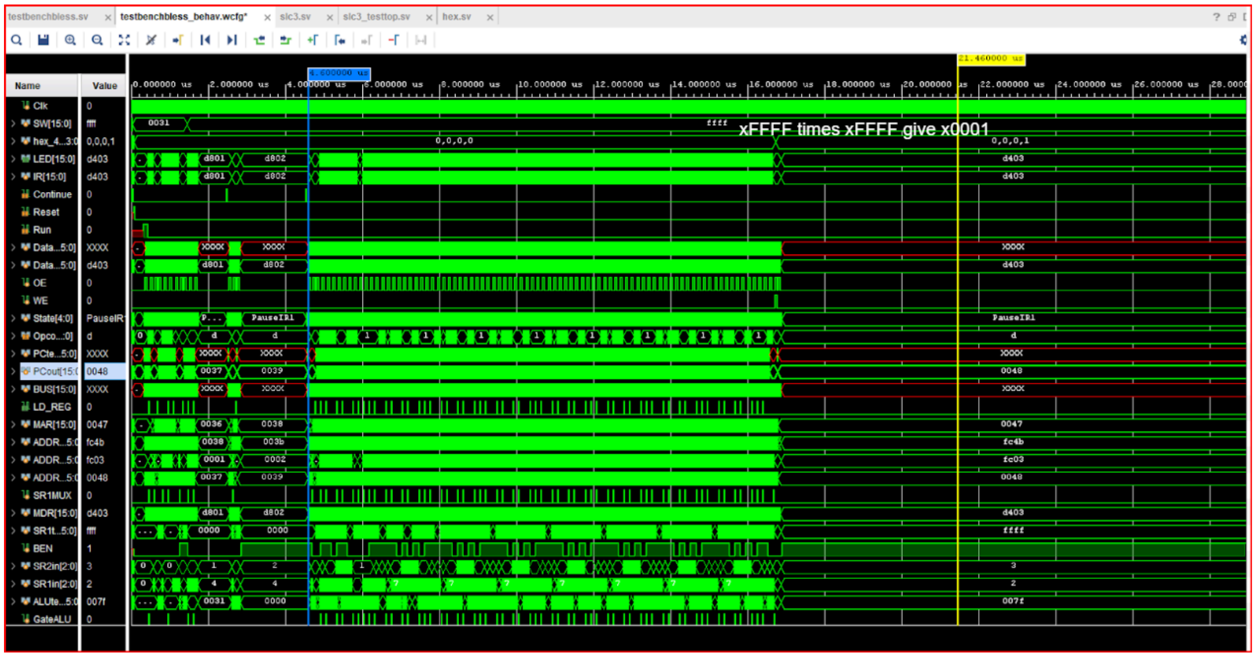

Figure 12: Multiplication Test Results

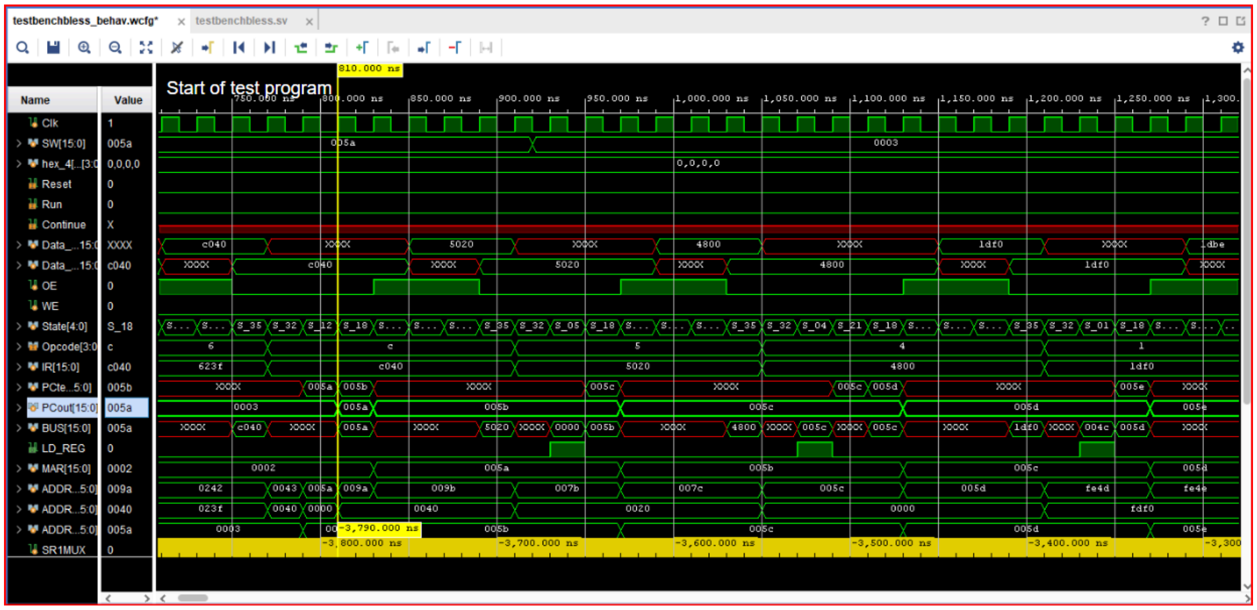

Figure 13: Start of sorting test programme

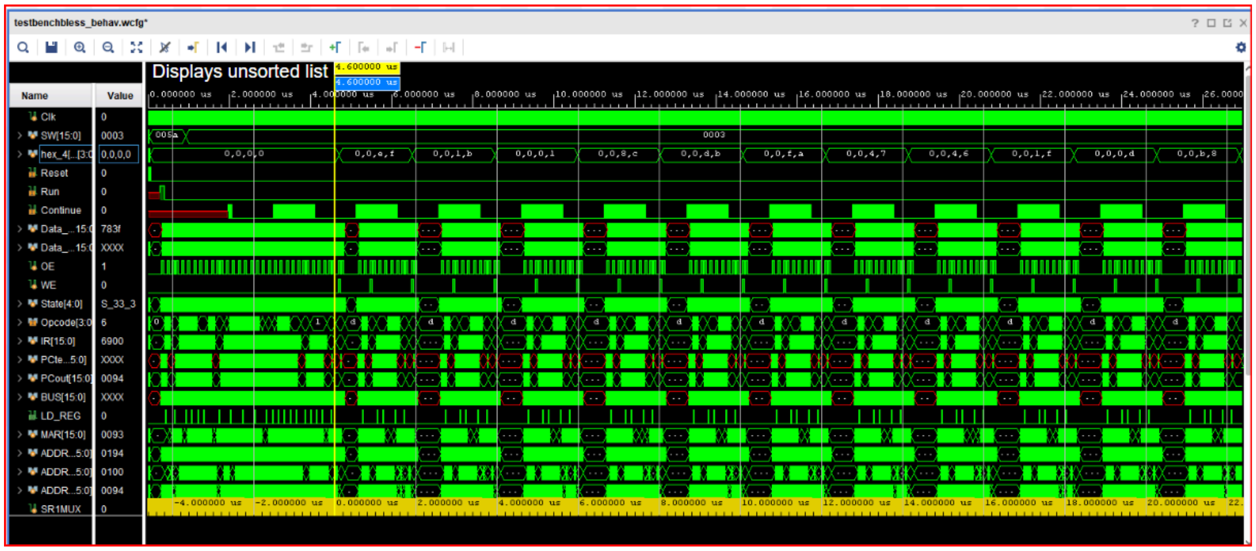

Figure 14: display function of unsorted list

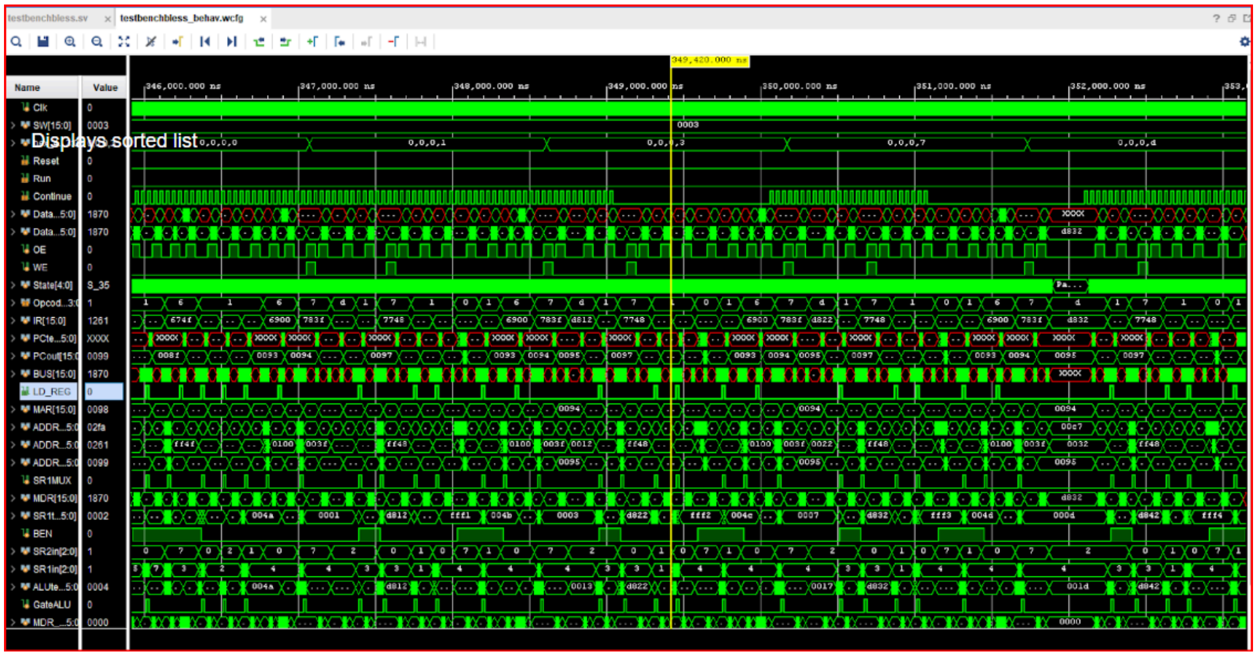

Figure 15 display function of sorted list

Post-Lab Questions

1) Design Resource and statistics Table: The frequency is calculated by doing 1/(10ns-wns).

LUT	2788
DSP	9
Memory (BRAM)	8
Flip-Flop	2592
Latches*	0
Frequency	0.105 MHz
Static Power	0.071 W
Dynamic Power	0.022 W
Total Power	0.093 W

2) What is MEM2IO used for, i.e. what is its main function?

MEM2IO is a memory mapped I/O that handles the inputs and outputs (switches and LEDs) based on the memory address loaded into the FSM. When the address is xFFFF we load the data from the switches into MDR and from SRAM otherwise. If WE is high and the address is xFFFF we write to the LEDs. We are also able to pass Data from CPU to the SRAM in the form of getting it from MDR.

3) What is the difference between BR and JMP instructions?

BR or branch operation allows for conditional checks for the N Z or P bit to be one before jumping to a different part of the program which is specified by a 9 bit value that is added to the current PC value.

However, JMP is an unconditional jump or branch that simply changes the PC location to a specific memory location that is stipulated by the value in the Base register that specified by the IR.

4) What is the purpose of the R signal in Patt and Patel?

The R signal is the ready bit signal used in the LC-3 instruction set processor to ensure that the memory has been completely loaded or completely stored before moving to another state or operation. This prevents memory from being loaded or stored with incomplete or incorrect values.

5) How do we compensate for the lack of the R signal in our design? What implications does this have for synchronization

Since we do not have an R signal in our design, we instead extend the FSM for the states which have an R signal in the original LC3 datapath into 3 states, this allows for enough time such that the MDR can be pulled to be stored at the memory location specified by the MAR or load with information stored at the address specified by the MAR.

This would mean that there would be no asynchronous inputs from an external device driver like the memory. This allows for the design to be fully synchronous, and there to be no timing issues between the memory and the finite state machine, or need for a synchroniser as used by the buttons.

Conclusion

In conclusion, we managed to build a working, simplified version of the Patt and Patel LC-3 ISA using Systemverilog, through a Moore Machine. The functionality of the circuit behaved as expected.

However, we encountered quite a few issues to troubleshoot. Many of these were to do with using the wrong signals in the relevant states. We also had an issue where the bubble sort program would end up displaying the IR after all the sorts were done, which turned out to be due to having inferred latches.

The given materials were good enough to perform this lab, but we believe a greater coverage on the already given modules such as the Mem2IO would have been much more helpful.