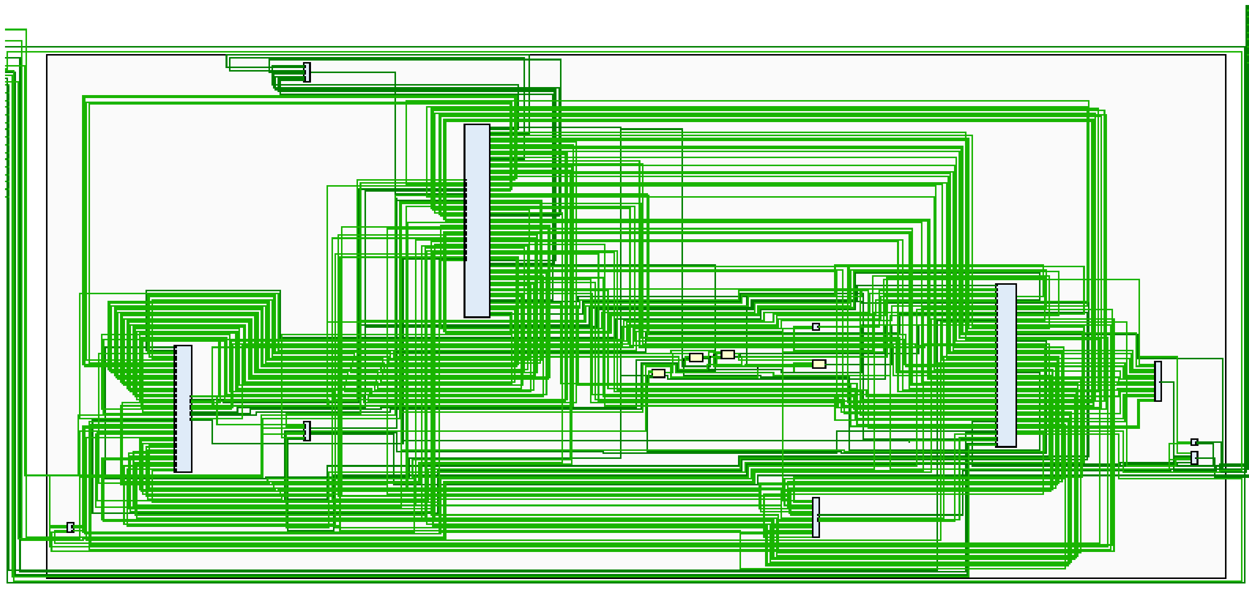
ECE 385

Fall 2023

Lab 5: SLC3 Computer

Alex Yu, Corey Yu
GL/ 13:45-14:00
Gene Lee

1. **IntroductionWritten**
    - The lab implements a simple microprocessor in SystemVerilog. It is a subset of the LC3 ISA from Patt & Patel with a 16 bit program counter, 16 bit instructions and 16 bit registers. The SLC3 has 11 instructions: 3 arithmetic ones (AND, NOT, ADD), 2 memory ones (LDR, STR), 3 jumps (BR, JSR, JMP) and a pause.
2. **Description and Diagrams of SLC**
    - **Summary of Operation**
        - **i.** The SLC3 consisted of PC, IR (instruction register), MAR (memory address register), MDR registers (memory data register), a 8x16 register module, ALU, status register and control unit. The control unit is a FSM that takes off the SLC3 processor that takes in Clk, Reset, Run, MBit and outputs ADD, SUB, SHIFTXAB, ClearXA, LoadB signals. The FSM has 29 states and fetches, decodes, and executes instructions.
    - **Describe in words how the SLC-3 performs its functions.**
        - i. The ISDU is a Moore FSM that controls the entire processor. The first three states are to fetch the instruction 18: PC is loaded into MAR and PC increments, 33: MDR is loaded with the memory content pointed to by MAR, 35: IR is loaded with MDR. The instruction is then decoded by state 32 which loads BEN and transitions to the appropriate execution state. The FSM then transitions through the necessary states to execute the operation.
    - c. Block Diagram of slc3.sv



    -
    - Written Description of all .sv modules
    - **Written Description of .sv Modules**

Module: slc3_sramtop.sv

input logic [15:0] SW, Clk, Reset,Run, Continue
output logic [15:0] LED, [7:0] hex_seg, hex_segB, [3:0] hex_grid, hex_gridB
Description: This is the top level file that controls the interaction between the physical on-chip memory, the slc3 file, and the other IO like LED, Hex Driver, and Switch.
Purpose: We need this file to connect the software signals to the hardware IO ports, and also read from the memory.

Module: slc3_testtop.sv

input logic [15:0] SW, Clk, Reset,Run, Continue
output logic [15:0] LED, [7:0] hex_seg, hex_segB, [3:0] hex_grid, hex_gridB
Description: This is a top level file that works exactly like sramtop.sv. The only difference is that this file doesn't actually output to the sram. It is a simulated sram used for simulation.
Purpose: We need this file for the simulation. When we are running the simulation, we can't access the actual sram, so we use the testtop instead.

Module: slc3.sv

input logic ([15:0] SW, Data_from_SRAM), Clk, Reset,Run, Continue
output logic ([15:0] LED, ADDR, Data_to_SRAM), ([7:0] hex_seg, hex_segB), ([3:0] hex_grid, hex_gridB), OE, WE
Description: This is the main file of this project. It calls all submodules and connects the signals between them. It implements the whole data path and is also responsible for communicating with the top level file, providing the signals needed for IOs and memories.
Purpose: We need this module to connect all the different parts of the circuit. Make sure all the inputs and outputs are coming from the right places. It is easy to make a small error in this module and break the whole SLC3 computer.

Module: Mem2IO.sv
input logic Clk, Reset, OE, WE, [15:0] ADDR, Switches, Data_from_CPU, Data_from_SRAM
output logic [15:0] Data_to_CPU, Data_to_SRAM, [3:0] HEX0, HEX1, HEX2, HEX3
output logic [3:0] HEX0, HEX1, HEX2, HEX3

Description: The mem2IO sits in-between the CPU, SRAM, Switches, and Displays. When WE is low and OE is high (a read state), if ADDR[15:0]==xFFFF then data_to_CPU is set to the switch register. When WE and OE are high, if ADDR[15:0]==xFFFF then hex_data is set to data_from_CPU.
Purpose: The module enables the memory mapped IO functions, it connects the switches and hex display.

Module: ISDU.sv
Input logic: input logic Clk, Reset, Run, Continue, input logic[3:0] Opcode, IR_5, IR_11, BEN
Output logic LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, GatePC, GateMDR, GateALU, GateMARMUX,
[1:0] PCMUX, DRMUX, SR1MUX, SR2MUX, ADDR1MUX,
[1:0] ADDR2MUX, ALUK,
Mem_OE, Mem_WE
Description: This is the control unit, it is a Moore FSM. There are 3 states for fetch, a decode state, and the necessary states for operations. In addition every state that accesses memory has 3 added wait states to ensure the memory read or write finishes. It generates the various load signals, gate signals for the bus, select signals e.g. SR1MUX, Mem_OE and WE.
Purpose: This controls the entire SLC3, the instructions are fetched, decoded, and executed according to the states of the FSM.

Module: reg_16.sv
Input logic Rst, Clk, [15:0] data_in, load,
Output logic [15:0] data_out
Description: Generic 16 bit register file that sets data_out to 0 if Rst is high, or sets data_out to data_in if load is high.
Purpose: This is the generic register file used to make PC, IR, MAR, MDR registers as well as the 8x16 register file.

Module: MUX.sv
Input logic: [15:0] data_in_1, [15:0] data_in_1,[15:0] data_in_1,[15:0] data_in_1,[15:0] data_in_1,[15:0] data_in_1,[15:0] data_in_1,[15:0] data_in_1, [2:0]select
Output logic: data_out
Description:This is a combined module for the muxes. This module includes the 4 to 1 mux, 2 to 1 mux, 8 to 1 mux, and etc. In this SLC3 implementation, there are multiple muxes we need to select the correct signal input and output.
Purpose: We need the muxes in multiple places, like the data bus is a mux that uses the gate signals to control the input.

Module: REG_FILE.sv
Input logic [15:0] data_in, Clk, Reset, [2:0] SR1_select, SR2_select, DR_select, LD_signal,
Output logic [15:0] SR1_out, SR2_out

Description: This is an 8x16 register file with two outputs SR1 and SR1 determined by SR1_select and SR2_select. LD_signal determines if some register is to be loaded and DR_select selects the register to be loaded.
Purpose: This is a register file that holds data that needs to be stored.

Module: ALU.sv
Input logic [15:0] A_input, [15:0] B_input, [1:0] select_signal,
Output logic [15:0] ALU_output
Description: This is an ALU that can perform and, addition, negation of two values. The particular operation is selected by the select signal: 00 corresponds to addition, 01 corresponds to and, 10 corresponds to negation and 11 corresponds to pass-through register A value.
Purpose: This is the arithmetic unit of the SLC3 that is used to perform operations

Module: NZP.sv
Input logic Clk, LD_CC, [15:0] data_in,
Output logic N_out, P_out, Z_out
Description: This is 3 one bit registers that when LD_CC is high takes in the 16 bit data_in, checks if the values is negative (n), zero (z) or positive (p) and stores the appropriate data into N Z and P
Purpose: This is set on operations that involve the ALU and LDR; N,Z,P are checked against the nzp in the branch opcode to determine whether to branch

Module: instantiateram.sv
Input Reset, Clk,
Output logic [15:0] ADDR, logic wren, [15:0] data
Description: This is a FSM that serves to instantiate the on chip memory with instructions
Purpose: To populate the on chip memory with the test programs so the instructions can be tested.

Module: test_memory.sv
Input Reset, Clk, [15:0] data, [9:0] address, ena, wren
Output logic [15:0] readout

Description: This simulates the behavior of an on-chip ram
Purpose: This program mimics the behavior of the on-chip memory so that you can run simulations (as on-chip memory obviously does not exist in sim)

Module: memory_contents.sv
No inputs or outputs
Description: This contains the instructions in the simulated test memory
Purpose: This is used in conjunction with test_memory.sv to simulate the behavior of on-chip memory to allow simulations

Module: SL3_2.sv
No inputs or outputs
Description: This defines all the instructions that SLC3 implements in terms of SystemVerilog functions
Purpose: Effectively an assembler that translates the instructions populated in memory into actual SystemVerilog so the fpga can understand it

Module: Module: Synchronizer.sv
Inputs: Clk, Reset, d
Outputs: q

Description: This acts like a flip flop that takes an asynchronous signal and makes it synchronous. The asynch input is fed into the flip flop and outputs the data on the next clock cycle.
Purpose: This module forces the entire design to be synchronous, or that every module runs on the same clock cycle.

Module: HexDriver.sv
Inputs: Clk, reset, [3:0] in[4], [3:0] nibble
Outputs: [7:0] hex_seg, [3:0] hex_grid, [7:0] hex

Description: The hex driver takes in the data in the binary format and outputs the number to hex that can be shown on the LED segment display.
Purpose: The LED segment display doesn’t take in numbers in binary so we need this module to translate it to display.

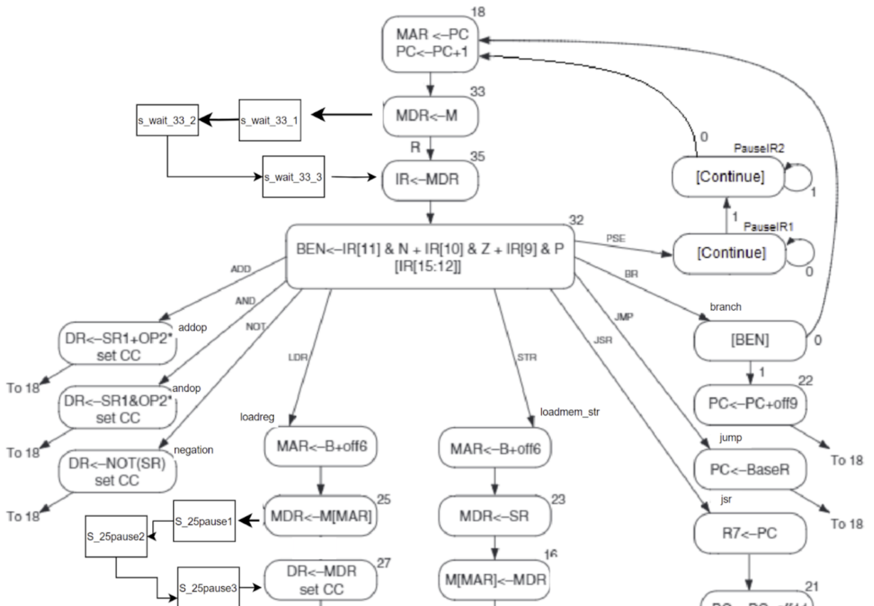
Module: Synchronizer.sv
Inputs: Clk, Reset, d
Outputs: q

Description: This acts like a flip flop that takes an asynchronous signal and makes it synchronous. The asynch input is fed into the flip flop and outputs the data on the next clock cycle.
Purpose: This module forces the entire design to be synchronous, or that every module runs on the same clock cycle.

- **f. Description of the operation of the ISDU (Instruction Sequence Decoder Unit) Named ISDU.sv, this is the control unit for the SLC-3. Describe in words how the ISDU controls the various components of the SLC-3 based on the current instruction.**

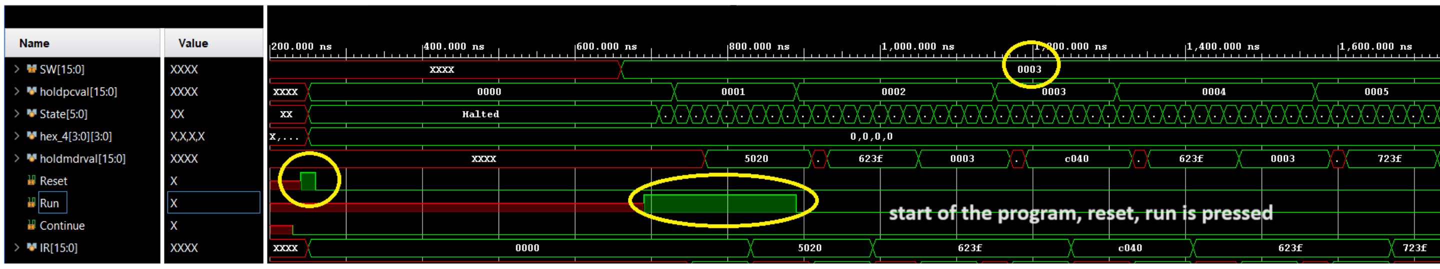
To demonstrate how the ISDU controls the SLC3, the control signals for each state in the ADD operation will be detailed. Default values for each control signal were assigned as 0 (or 00.. depending on the bits) before the cases to avoid an inferred latch. In state 18: MAR is loaded with PC value and PC increments. As such GatePC = 1'b1 (as PC must go through the bus), LD_MAR = 1'b1, PCMUX = 2'b00 (to select the increment option) and LD_PC = 1'b1. Then MDR is loaded with M[MAR] the memory pointed to by MAR. Mem_OE = 1'b1 (to enable a read) and LD_MDR = 1'b1. This is the case for the 3 wait states following this as well. Note that technically only 2 additional wait states were needed (as it is net 3 states not 3 more to ensure enough time for memory read/write to complete). In state 35: IR is loaded with MDR; control signals are GateMDR = 1'b1, LD_IR = 1'b1. This concludes the fetch portion. Then the instruction is decoded in state 32 and BEN is loaded, LD_BEN=1’b1. Then the instruction is executed in the addop state. The control signals were set to: SR2MUX = IR_5 (to select between choosing adding an immediate value or another register), ALUK = 2'b00 (to choose addition), GateALU = 1'b1 (as the resulting data has to pass through the bus) , LD_REG = 1'b1 (to load the result into the destination register), SR1MUX = 1'b1 (which corresponds to bits 8:6 of IR), DRMUX = 1'b0 (which corresponds to bits 11:9 of IR) and LD_CC = 1 to load the NZP register with the appropriate values.

- **g. State Diagram of ISDU**



3. Simulations of SLC-3 Instructions

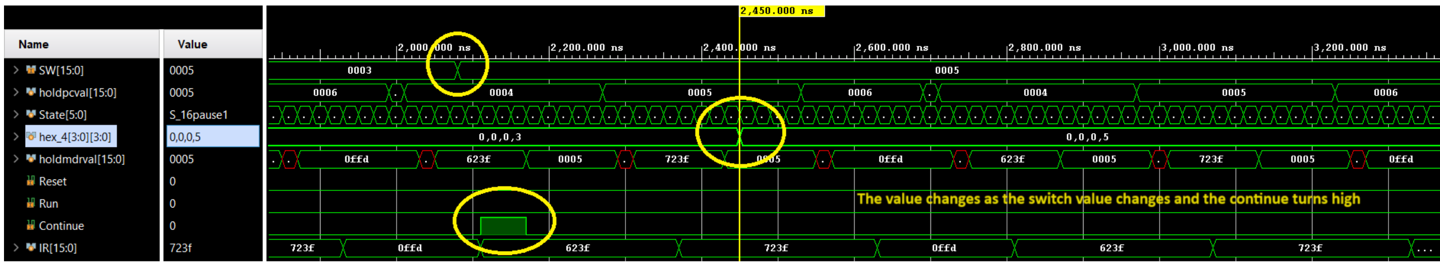
I/O Test 1



Name	Value
SW[15:0]	XXXX
holdpcval[15:0]	XXXX
State[5:0]	XX
hex_4[3:0][3:0]	X,X,X,X
holdmdrval[15:0]	XXXX
Reset	X
Run	X
Continue	X
IR[15:0]	XXXX

start of the program, reset, run is pressed

Start of I/O Test 1, the switch is set to 0003, Reset, and Run are pressed.

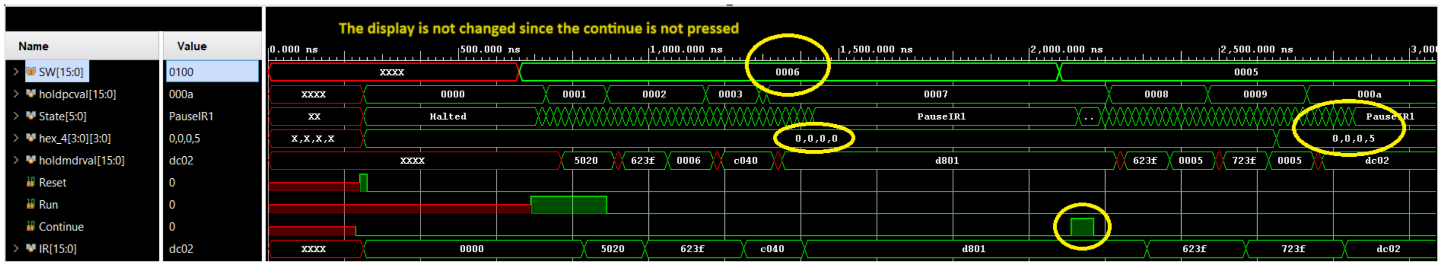


Name	Value
SW[15:0]	0005
holdpcval[15:0]	0005
State[5:0]	S_16pause1
hex_4[3:0][3:0]	0,0,0,5
holdmdrval[15:0]	0005
Reset	0
Run	0
Continue	0
IR[15:0]	723f

The value changes as the switch value changes and the continue turns high

In the middle of I/O Test 1, the value changes with the switch

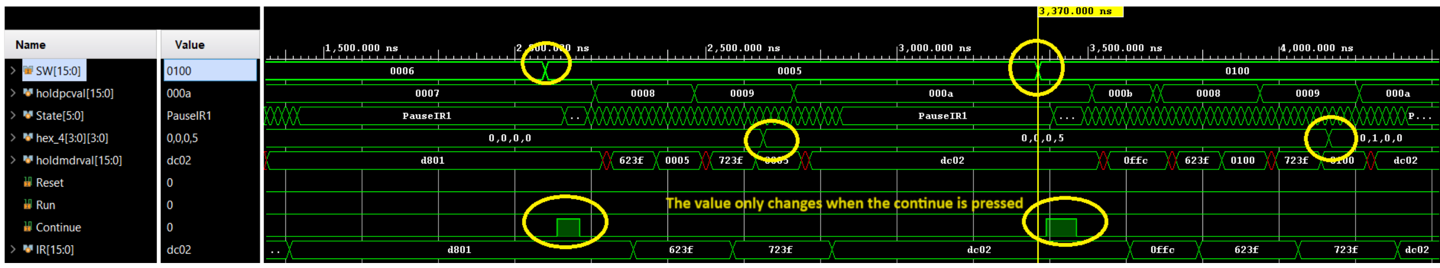
I/O Test 2



The display is not changed since the continue is not pressed

Name	Value
SW[15:0]	0100
holdpcval[15:0]	000a
State[5:0]	PauseIR1
hex_4[3:0][3:0]	0,0,0,5
holdmdrval[15:0]	dc02
Reset	0
Run	0
Continue	0
IR[15:0]	dc02

Start of I/O Test 2, the display value doesn’t change because the continue button is not pressed

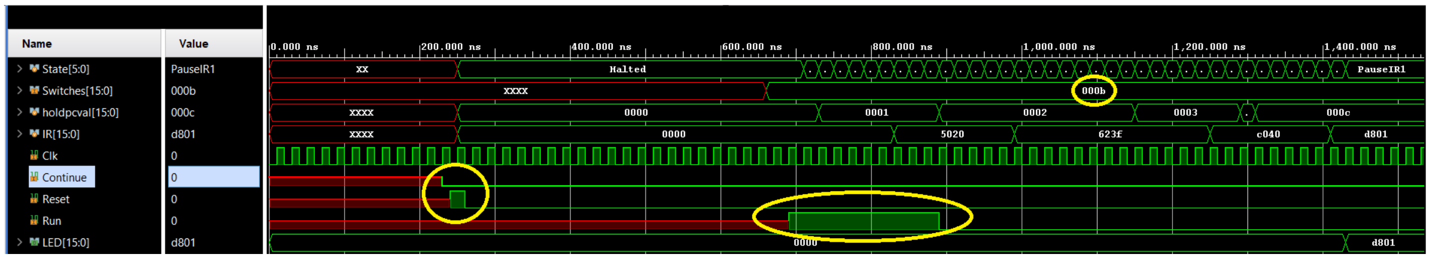


Name	Value
SW[15:0]	0100
holdpcval[15:0]	000a
State[5:0]	PauseIR1
hex_4[3:0][3:0]	0,0,0,5
holdmdrval[15:0]	dc02
Reset	0
Run	0
Continue	0
IR[15:0]	dc02

The value only changes when the continue is pressed

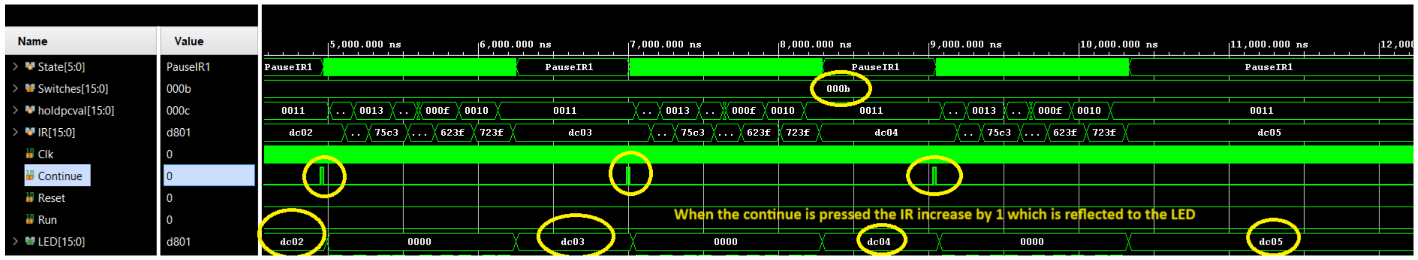
The display value changes after the continue button is pressed

Self modifying code Test



Name	Value
State[5:0]	PauseIR1
Switches[15:0]	000b
holdpcval[15:0]	000c
IR[15:0]	d801
Clk	0
Continue	0
Reset	0
Run	0
LED[15:0]	d801

The start of the self-modifying code test, the switch is set to 000B, Reset, and Run are pressed.

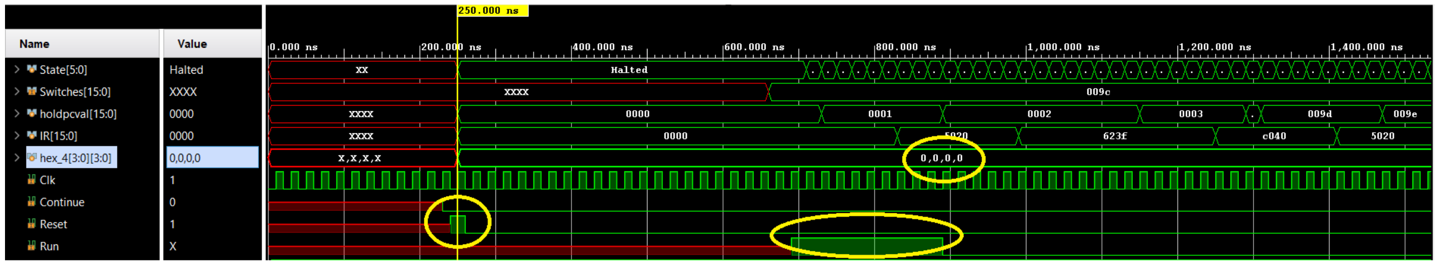


Name	Value
State[5:0]	PauseIR1
Switches[15:0]	000b
holdpcval[15:0]	000c
IR[15:0]	d801
Clk	0
Continue	0
Reset	0
Run	0
LED[15:0]	d801

When the continue is pressed the IR increase by 1 which is reflected to the LED

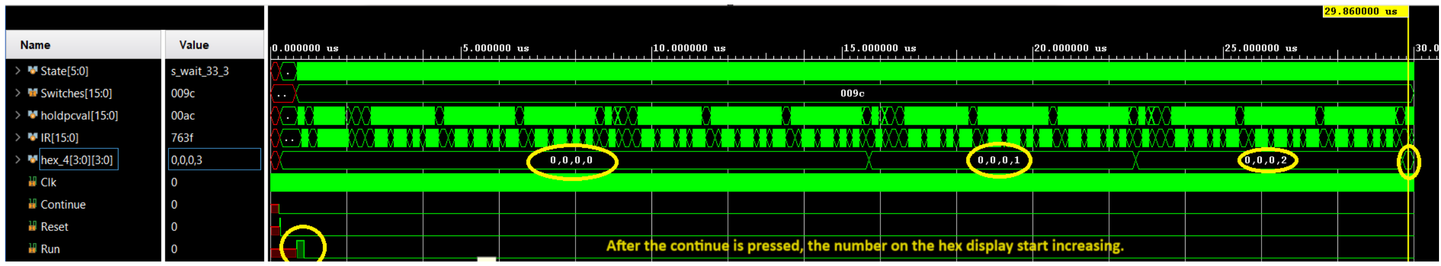
When the continue is pressed the IR increase by 1 which is reflected to the LED

Auto Counting Test



Name	Value
State[5:0]	Halted
Switches[15:0]	XXXX
holdpcval[15:0]	0000
IR[15:0]	0000
hex_4[3:0][3:0]	0,0,0,0
Clk	1
Continue	0
Reset	1
Run	X

The start of the auto counting test, the switch is set to 009C, Reset, and Run are pressed.

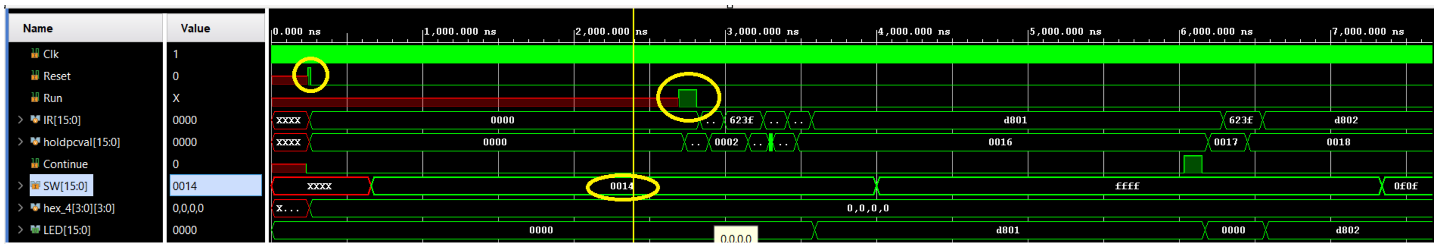


Name	Value
State[5:0]	s_wait_33_3
Switches[15:0]	009c
holdpcval[15:0]	00ac
IR[15:0]	763f
hex_4[3:0][3:0]	0,0,0,3
Clk	0
Continue	0
Reset	0
Run	0

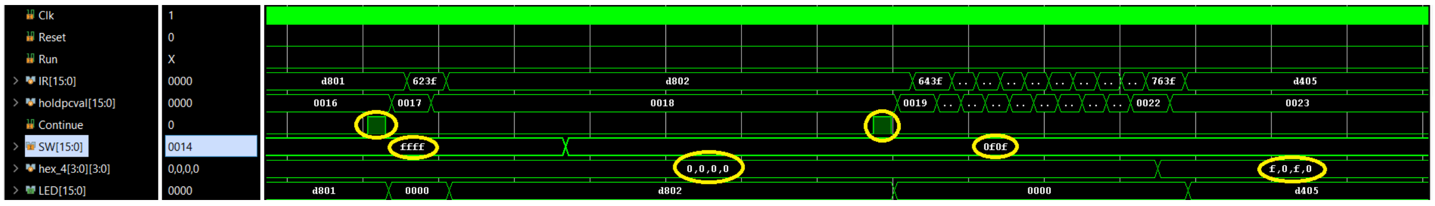
After the continue is pressed, the number on the hex display start increasing.

After the continue is pressed, the number on the hex display starts increasing.

XOR Test

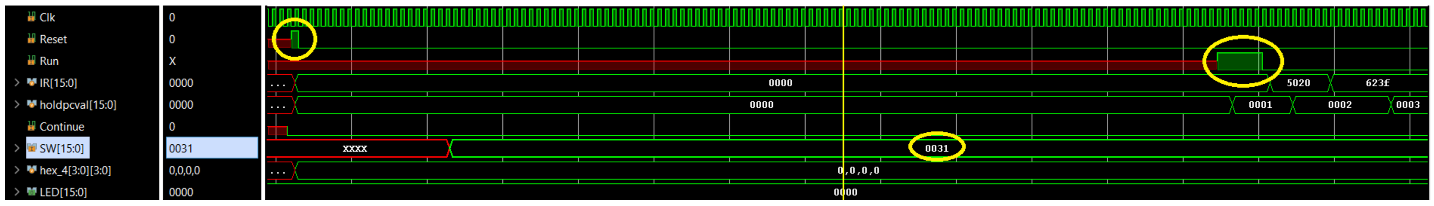


Start of the XOR, switch is set to 0014, Reset, and Run are pressed.

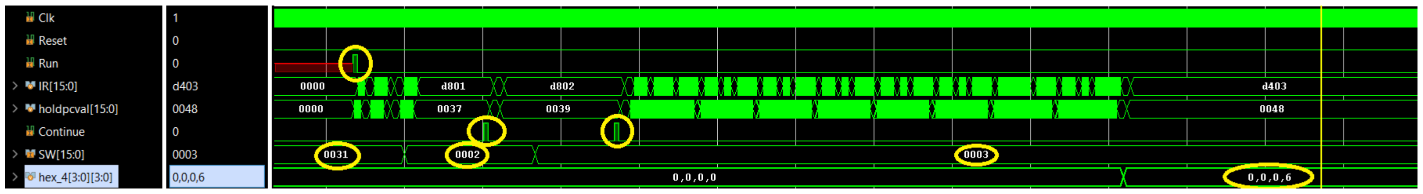


After loading the switch to xFFFF, press continue, and load the switch to x0F0F, press continue again. We can see the output value be displayed as xF0F0.

Multiplication Test

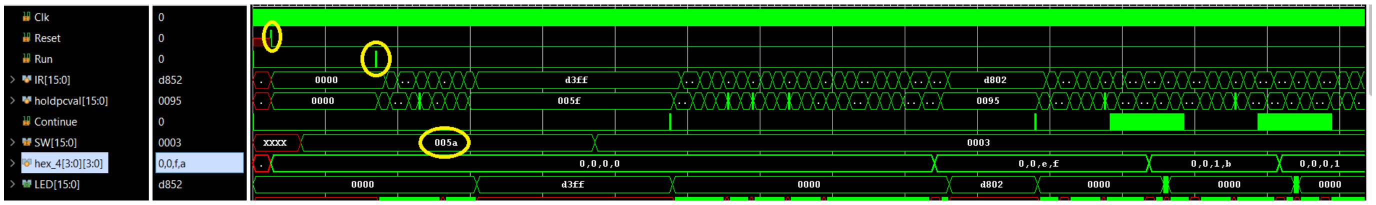


Start of the Multiplication, switch is set to 0031, Reset, and Run are pressed.

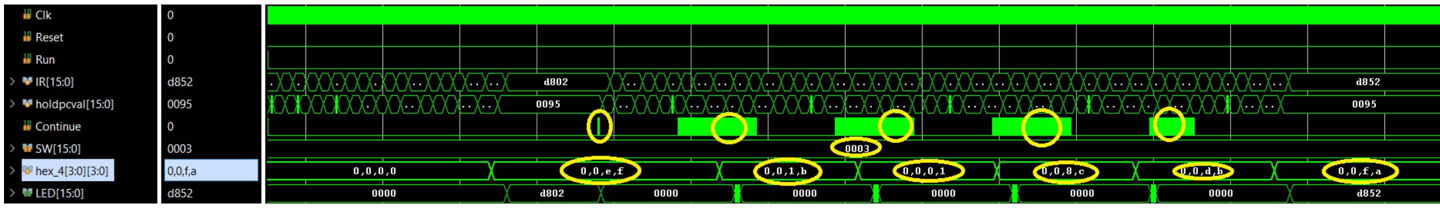


After loading the switch to x0002, press continue, and load the switch to x0003, press continue again. We can see the output value be displayed as x0006.

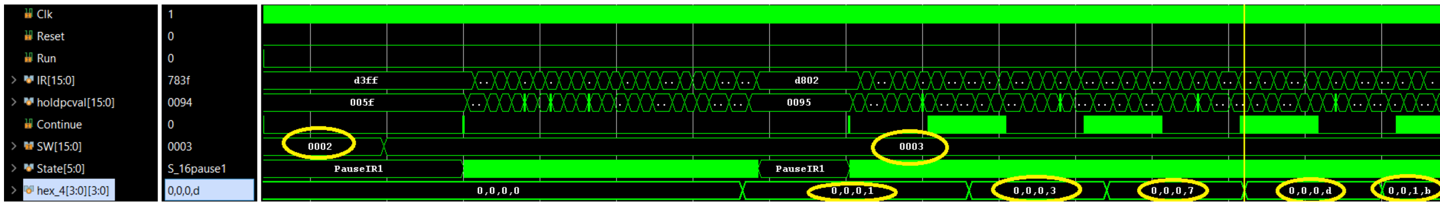
Sort Test



Start of the SORT test, the switch is set to 005A, Reset, and Run are pressed.



First, we turn the switch to 3 to see the list before sorted. By pressing continue, we were able to iterate through the list.

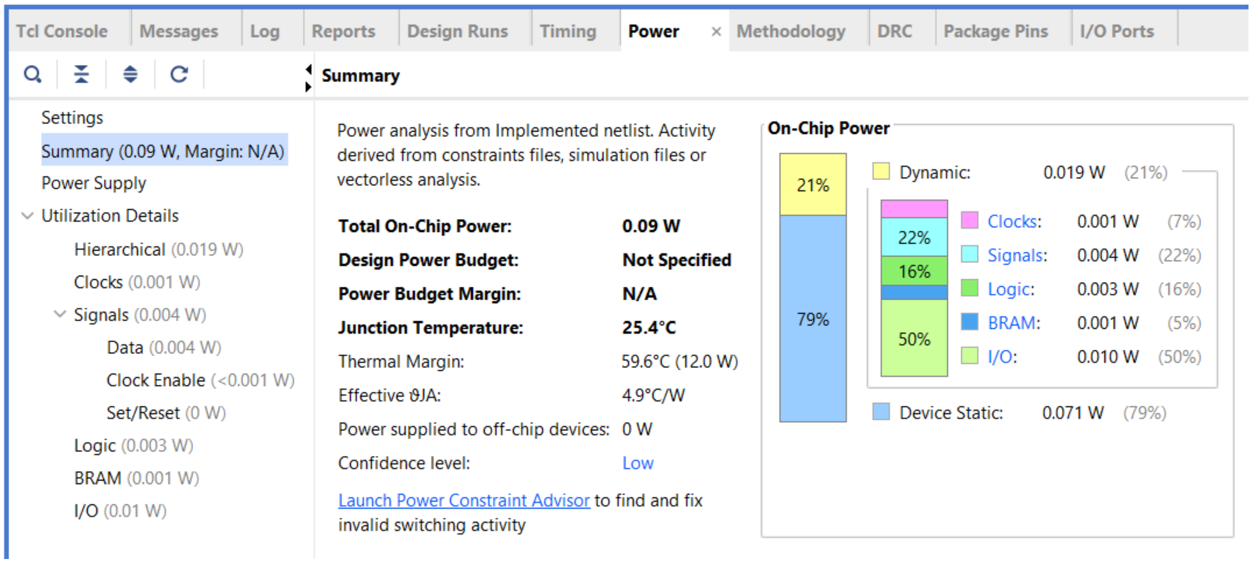


After that, we change the switch to 2 to let it sort the list. Now, if we change the switch to 3 to view the list, the list is sorted.

4. Post-Lab Questions

	SLC3
LUT	**464**
DSP	**0**
Memory (BRAM)	**0**
Flip Flop	**262**
Frequency	**0.1037**
Static Power	**0.071W**
Dynamic Power	**0.019W**

Total Power	**0.09W**



Settings
Summary (0.09 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (0.019 W)
Clocks (0.001 W)
Signals (0.004 W)
Data (0.004 W)
Clock Enable (<0.001 W)
Set/Reset (0 W)
Logic (0.003 W)
BRAM (0.001 W)
I/O (0.01 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.09 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	59.6°C (12.0 W)
Effective ϑJA:	4.9°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	0.019 W	(21%)
Clocks:	0.001 W	(7%)
Signals:	0.004 W	(22%)
Logic:	0.003 W	(16%)
BRAM:	0.001 W	(5%)
I/O:	0.010 W	(50%)
Device Static:	0.071 W	(79%)

- **What is MEM2IO used for, i.e. what is its main function?**

MEM2IO is a memory mapped io module. When the switches are set to a value, this data is attempted to be loaded into memory location xFFFF. MEM2IO intercepts this and pushes the data into the MDR register. When data is stored into xFFFF, MEM2IO pushes the data to the hex drivers (and the displays).

- **What is the difference between BR and JMP instructions?**

BR is branch, can be conditional, and has a jump of 9 bits, meaning it can store up to [0, 00FF] and FFFF and that to PC meaning you can only go to a range of values. JUMP is unconditional and sets the PC to the register value, meaning you can jump to anywhere.

- **What is the purpose of the R signal in Patt and Patel? How do we compensate for the lack of the signal in our design? What implications does this have for synchronization?**

The R signal is generated by memory in P&P when it is done reading or writing (a ready signal). The memory instantiated on the FPGA in this lab does not have a ready signal; to compensate for this 3 wait states were added to ensure the operation involving memory has finished. Rather than waiting for an external R signal, the wait for a memory write or read is a fixed 4 states, forcing these to be synchronous.

5. **Document any problems you encountered and your solutions to them, and a short conclusion.**

In this lab, a simplified version of the LC3 processor was implemented. It is capable of performing a variety of operations: arithmetic/logical, loading from/storing into memory and jumping to various points of the program. A large source of errors were missing a control signal for a particular state; this was fixed by rechecking through the expected control signals for every state in the ISDU. We also chose to name the first state of each operation by name rather than number to assist with debugging. This was the first complicated lab in the class, and yet for our group it was the one we finished the earliest as we started the weekend beforehand.