Simple Computer SLC-3.2 in SystemVerilog

Hongbo Zheng, Yuhao Yuan

Summer 2021

ECE 385

Section AB1

TA: Hanfei Wang

Introduction

The purpose of this lab is to design a simple microprocessor called SLC-3 using SystemVerilog. The microprocessor is a simplified version of LC-3 ISA which is able to read the 16-bit instruction stored at a specific address inside the memory, then decode the 16-bit instruction based on the first 4-bit opcode of the instruction, and finally execute the 16-bit instruction. Overall, the SLC-3 processor is capable of reading from the memory, writing to the memory, changing PC (program counter) value, handling different status (nzp), doing addition, and performing simple logic operations: AND, NOT.

Written Description of the Circuit

Summary of operation

The SLC-3 processor performs operations in three steps: Fetch, Decode, and Execute. In Fetch state, the SLC-3 processor uses the 16-bit value stored inside the PC register as the address to read from the memory. The 16-bit PC value first gets passed into the MAR (Memory Address Register). Then, MAR passes the 16-bit address to the memory, and the data located at this specific 16-bit address inside memory is stored into MDR (Memory Data Register). Finally, the IR (Instruction Register) receives the memory data from MDR, and the PC value gets incremented by 1 in order to read the data stored at the next address.

In Decode state, the first 4-bit opcode of the 16-bit value stored inside IR is used to determine which instruction to perform. Part of the rest 12-bit can possibly be the 3-bit DR (Destination Register), 3-bit SR1 (Source Register 1), 3-bit SR2 (Source Register 2), 3-bit BaseR (Base Register), 9-bit/11-bit PC offset, 6-bit offset for memory address, 5-bit immediate value for AND and ADD operation, and 12-bit LED for Pause, depended on which instruction the SLC-3 processor is going to perform.

In Execute state, the SLC-3 processor will fetch operands and execute the operation. Then, the result will be stored into some places, such as the memory, PC register, or 1 of the 8 registers inside the register file.

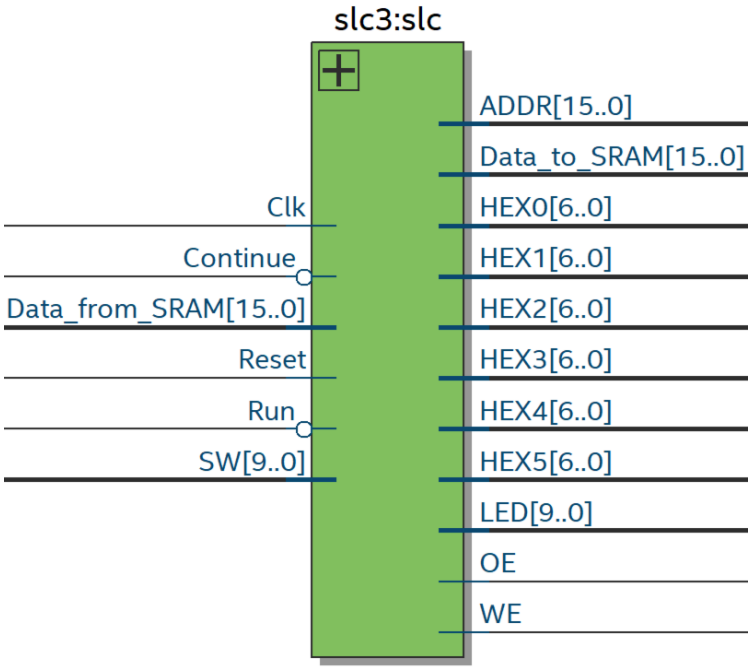
The SLC-3 processor is capable of performing 1 of the 11 instructions: ADD, ADDi, AND, ANDi, NOT, BR, JMP, JSR, LDR, STR, PAUSE, during each Fetch-Decode-Execute cycle. The detailed Instruction Summary is shown in the table below.

Instruction	Instruction [15:0]										Operation
ADD	0	0	0	1	DR	SR1	0	0	0	SR2	R(DR) ← R(SR1) + R(SR2)
	16-bit value stored in SR1 is added with 16-bit value stored in SR2, and the result is stored into DR. Set the status register.										
ADDi	0	0	0	1	DR	SR	1	imm5			R(DR) ← R(SR) + SEXT(imm5)
	16-bit value stored in SR is added with 16-bit sign-extended imm5, and the result is stored into DR. Set the status register.										
AND	0	1	0	1	DR	SR1	0	0	0	SR2	R(DR) ← R(SR1) AND R(SR2)
	16-bit value stored in SR1 is AND with 16-bit value stored in SR2, and the result is stored into DR. Set the status register.										

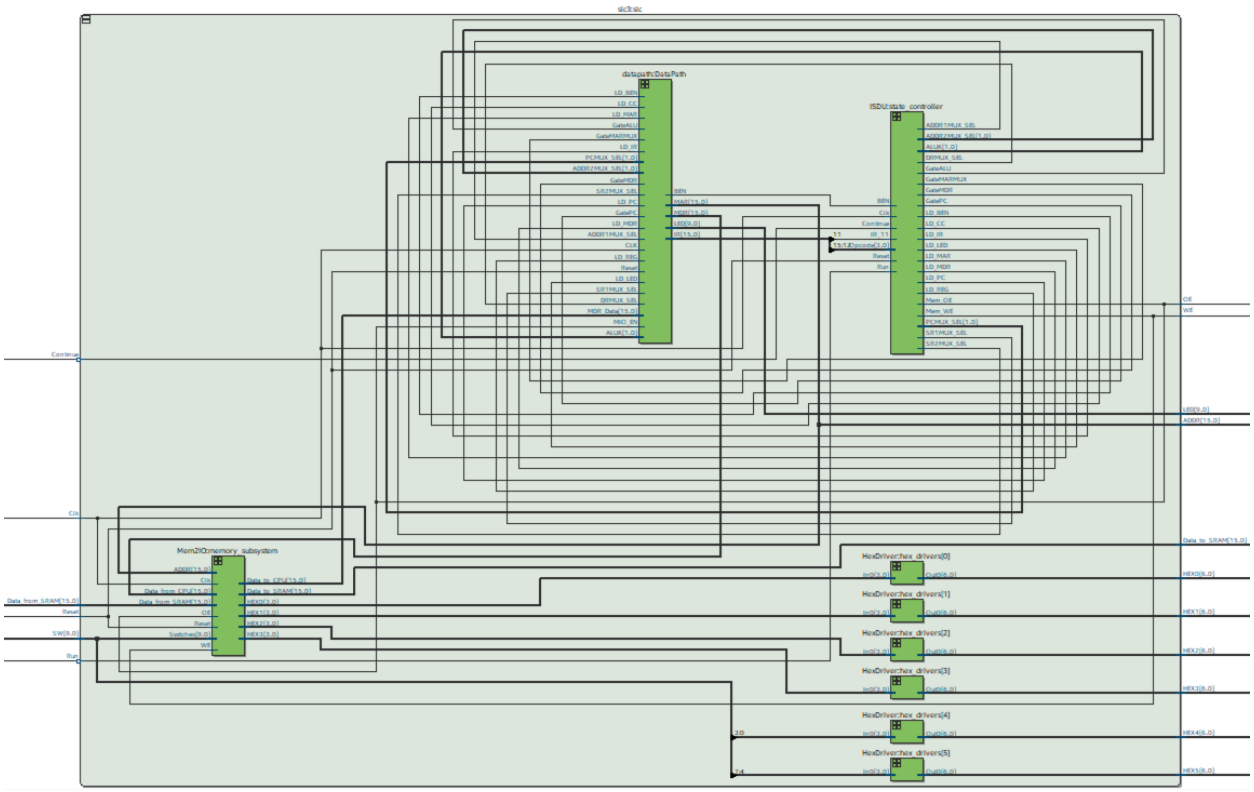
ANDi	0	1	0	1	DR		SR	1	imm5					R(DR) ← R(SR) AND SEXT(imm5)
	16-bit value stored in SR is AND with 16-bit sign-extended imm5, and the result is stored into DR. Set the status register.													
NOT	1	0	0	1	DR		SR	1	1	1	1	1	1	R(DR) ← NOT R(SR)
	16-bit value stored in SR is inverted (NOT), and the result is stored into DR. Set the status register.													
BR	0	0	0	0	n	z	p	PCoffset11						if ((nzp AND NZP) != 0) PC ← PC + SEXT(PCoffset9)
	If any of nzp value in the instruction is matched with the NZP stored in the status register, take the branch (add PCoffset9 to PC); otherwise continue execution.													
JMP	1	1	0	0	0	0	0	BaseR	offset6					PC ← R(BaseR)
	Jump. Copy the memory address from BaseR to PC register.													
JSR	0	1	0	0	1	PCoffset11								R(7) ← PC PC ← PC + SEXT(PCoffset11)
	Jump to Subroutine. Store current 16-bit PC value into R7, then add PCoffset11 to current PC value.													
LDR	0	1	1	0	DR		BaseR	offset6						R(DR) ← M[R(BaseR) + SEXT(offset6)]
	Load DR with the memory contents at the address BaseR+SEXT(offset6). Set the status register.													
STR	0	1	1	1	SR		BaseR	offset6						M[R(BaseR) + SEXT(offset6)] ← R(SR)
	Store the contents of SR in the memory at the address BaseR+SEXT(offset6).													
PAUSE	1	1	0	1	ledVect12									LED ← ledVect12; Wait on Continue
	Pause execution until Continue is asserted by the user. While paused, ledVect12 is displayed on the board LEDs.													

Block Diagrams and Module Descriptions

slc3 High-Level Block Diagram



slc3 Block Diagram



Module Description

Module: slc3.sv

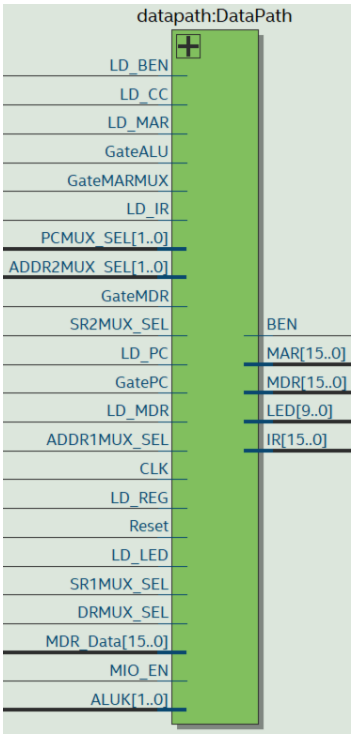
Input: Clk, Reset, Run, Continue, [9:0] SW, [15:0] Data_from_SRAM

Output: OE, WE, [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, [6:0] HEX3, [6:0] HEX4, [6:0] HEX5, [9:0] LED, [15:0] ADDR, [15:0] Data_to_SRAM

Description: This component contains the datapath, ISDU (Instruction Sequencer Decoder Unit), Mem2IO, and HexDrivers. The datapath and the ISDU together form the central processing unit (CPU), the Mem2IO manages all the I/O with the DE10-Lite physical I/O devices: switches and 7-segment display, and the HexDrivers are used to display the hex-value of the 16-bit memory content and the 8-bit switch values. When Run and Continue are both low, the SLC-3 resets to its initial address: x0000, and its initial state: Halted. When Run is low, the SLC-3 processor starts the Fetch-Decode-Execute cycle. Continue is used to start another Fetch-Decode-Execute cycle when the state machine is in Pause states. [9:0] SW value is used to get the input from the user. [15:0] Data_from_SRAM is the 16-bit data reading from the memory. OE and WE are outputs from ISDU which controls reading from the memory or writing into the memory. [6:0] HEX is used to display content on the HEX unit. [9:0] LED is used to display the lower 10-bit of ledVect12 for Pause instruction on the DE10-Lite board. [15:0] ADDR is the 16-bit address passing to the SRAM. [15:0] Data_to_SRAM is the 16-bit data writing into the memory.

Purpose: This module is used as the high-level block of the SLC-3 processor without the memory component.

datapath High-Level Block Diagram



Module Description

Module: datapath.sv

Input: CLK, Reset, GatePC, GateMARMUX, GateMDR, GateALU, LD_REG, LC_PC, LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_LED, DRMUX_SEL, SR1MUX_SEL, SR2MUX_SEL, ADDR1MUX_SEL, MIO_EN, [1:0] ALUK, [1:0] PCMUX_SEL, [1:0] ADDR2MUX_SEL, [15:0] MDR_Data

Output: [15:0] MAR, [15:0] MDR, [15:0] IR, [9:0] LED

Description: This component contains registers: PC, MAR, MDR, IR, R0-R7 (in REG_FILE), BEN, n_out, z_out, p_out, LED, multiplexers: DRMUX, SR1MUX, SR2MUX, PCMUX, ADDR1MUX, ADDR2MUX, MDRMUX, DataBus_Mux, 16-bit databus, and some combinational logics for instruction decoding, instruction processing, data processing, and data transferring in SLC-3 processor. CLK and Reset are connected to all the registers. When Reset is high, all the registers are cleared (store the binary value 0) on the positive edge of CLK. GateXXXs are used as the select signals for 16-bit databus. The LD_XXs controls the data-loading of corresponding registers. XXX_SELs controls the select signals of corresponding multiplexers. MIO_EN chooses the input of the MDR. [1:0] ALUK controls which operation to perform by the ALU. [15:0] MDR_Data is the 16-bit data from the memory.

Purpose: This module is used as the datapath of the SLC-3 processor which is mostly consists of multiplexers and registers. The module is mainly used for instruction decoding, instruction processing, data processing, and data transferring.

DR_SEL	DRMUX Output
0	IR[11:9]
1	3'b111

SR1_SEL	SR1MUX Output
0	IR[11:9]
1	IR[8:6]

SR2_SEL	SR2MUX Output
0	SR2_OUT
1	16-bit SEXT IR[4:0]

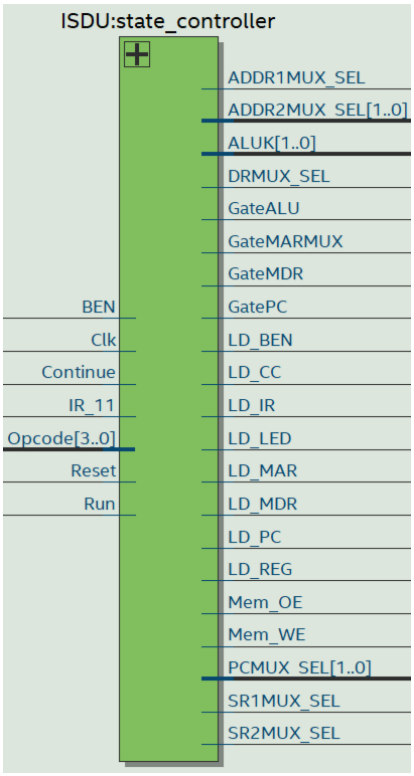
PC_SEL	PCMUX Output
00	PC+1
01	ADDR_SUM
10	DataBus
11	16'h0

ADDR1_SEL	ADDR1MUX Output
0	SR1_OUT
1	PC

ADDR2_SEL	ADDR2MUX Output
00	16'h0
01	16-bit SEXT IR[5:0]
10	16-bit SEXT IR[8:0]
11	16-bit SEXT IR[10:0]

DataBus_SEL	DataBus Output
GatePC = 1'b1	PC
GateMARMUX =1'b1	ADDR_SUM
GateMDR = 1'b1	MDR
GateALU = 1'b1	ALU
ELSE	16'h0

ISDU High-Level Block Diagram



Module Description

Module: ISDU.sv

Input: Clk, Reset, Run, Continue, [3:0] Opcode, IR_11, BEN

Output: LD_MAR, LD_MDR, LD_IR, LD_BEN, LD_CC, LD_REG, LD_PC, LD_LED, GatePC, GateMARMUX, GateMDR, GateALU, DRMUX_SEL, SR1MUX_SEL, SR2MUX_SEL, ADDR1MUX_SEL, [1:0] PCMUX_SEL, [1:0] ADDR2MUX_SEL, [1:0] ALUK, Mem_OE, Mem_WE

Description: This component controls the state transitions of the SLC-3 processor and send out the correct control outputs to the datapath in each state to ensure the processor is performing the correct operation. The FSM will first execute the Fetch state which has 3 parts. It first passes the address from PC to MAR, and then the memory content at that location will be loaded into MDR. After that, IR will receive the instruction from MDR. Then, the FSM will perform the second state which is the Decode state. The state machine will choose the next state based on the first 4-bit opcode of the instruction stored in IR. Finally, the state machine will execute the all the states which are required for that specific instruction and then go back to Fetch state. The FSM may need to execute one or several states in Execute state. In each state, there are specific control outputs to the datapath to make sure that the correct paths are activated to process and transfer data.

[3:0] Opcode is the first 4-bit of the instruction stored in IR. IR_11 is the 11^{th} bit of the instruction. BEN is the output of the BEN register to check if the SLC-3 processor needs to perform a BR instruction. When Reset is high, the FSM is reset to its initial state: Halted. When Run is high, the FSM starts to perform the Fetch-Decode-Execute cycle. When the FSM is in Pause states, Continue needs to be high to allow the FSM to start another Fetch-Decode-Execute cycle.

Purpose: This module is used as the control unit of the SLC-3 processor which controls the state transitions of the SLC-3 processor based on the first 4-bit opcode of the instruction. In each state, the ISDU ensures the correct outputs are sent out to the datapath in order to activate the right path to process data. Therefore, the datapath is able to choose the output from the correct register, the correct register to load data, the correct input of multiplexer, the correct output of DataBus, the correct operation of ALU, and the correct mode of memory (Read or Write).

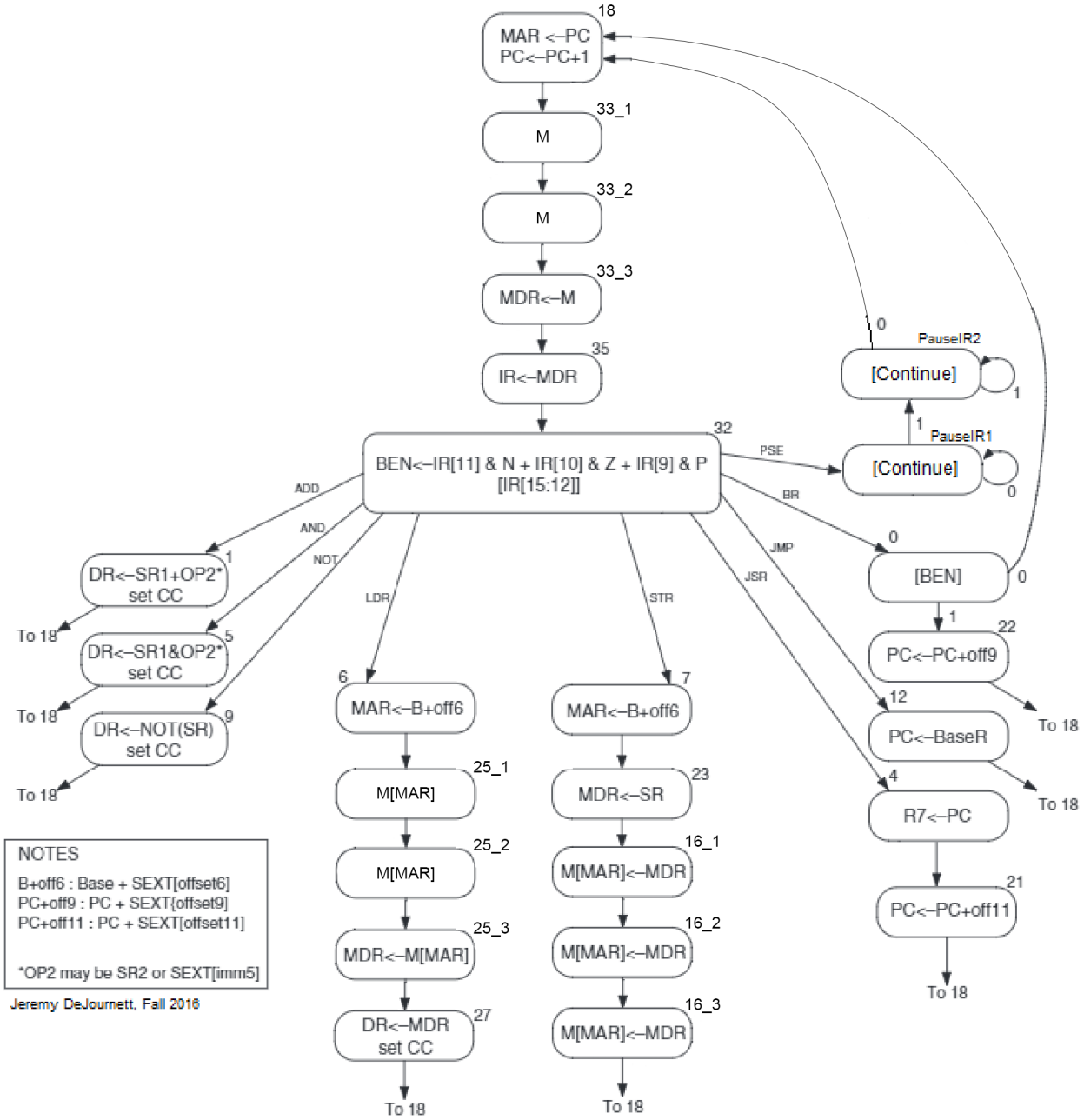
Instruction	FSM State	Control Outputs	Next State
Fetch	S_18 S_33_3	GatePC = 1'b1; LD_MAR = 1'b1; PCMUX_SEL = 2'b00; LD_PC = 1'b1;	S_33_1
	S_33_1 S_33_2	Mem_OE = 1'b1;	S_33_3
	S_33_3	Mem_OE = 1'b1; LD_MAR = 1'b1;	S_35
	S_35	GateMDR = 1'b1; LD_IR = 1'b1;	S_32
Decode	S_32	-----	Depend on Opcode

Instruction	Opcode	FSM State	Control Outputs	Next State
ADD / ADDi	0001	S_01	DRMUX_SEL = 1'b0; SR1MUX_SEL = 1'b1; ALUK = 2'b00; GateALU = 1'b1; LD_REG = 1'b1; LD_CC = 1'b1;	S_18
AND / ANDi	0101	S_05	DRMUX_SEL = 1'b0; SR1MUX_SEL = 1'b1; ALUK = 2'b01; GateALU = 1'b1; LD_REG = 1'b1; LD_CC = 1'b1;	S_18

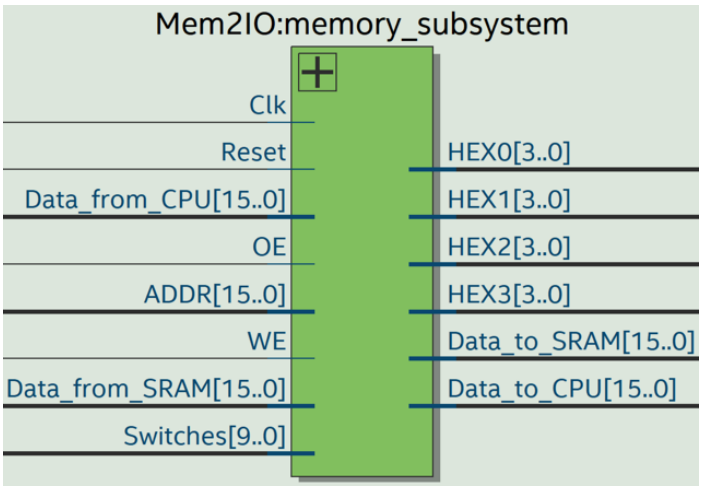
NOT	1001	S_09	DRMUX_SEL = 1'b0; SR1MUX_SEL = 1'b1; ALUK = 2'b10; GateALU = 1'b1; LD_REG = 1'b1; LD_CC = 1'b1;	S_18
BR BEN=1'b0	0000	-----	-----	S_18
BR BEN=1'b1	0000	S_22	ADDR1MUX_SEL = 1'b1; ADDR2MUX_SEL = 2'b10; PCMUX_SEL = 2'b01; LD_PC = 1'b1;	S_18
JMP	1100	S_12	SR1MUX_SEL = 1'b1; ADDR1MUX_SEL = 1'b0; ADDR2MUX_SEL = 2'b00; PCMUX_SEL = 2'b01; LD_PC = 1'b1;	S_18
JSR	0100	S_04	DRMUX_SEL = IR_11; GatePC = 1'b1; LD_REG = 1'b1;	S_21
		S_21	ADDR1MUX_SEL = 1'b1; ADDR2MUX_SEL = 2'b11; PCMUX_SEL = 2'b01; LD_PC = 1'b1;	S_18
LDR	0110	S_06	SR1MUX_SEL = 1'b1; ADDR1MUX_SEL = 1'b0; ADDR2MUX_SEL = 2'b01; GateMARMUX = 1'b1; LD_MAR = 1'b1;	S_25_1
		S_25_1 S_25_2	Mem_OE = 1'b1; Mem_WE = 1'b0;	S_25_3
		S_25_3	Mem_OE = 1'b1; Mem_WE = 1'b0; LD_MDR = 1'b1;	S_27
		S_27	GateMDR = 1'b1; DRMUX_SEL = 1'b0; LD_REG = 1'b1; LD_CC = 1'b1;	S_18
STR	0111	S_07	SR1MUX_SEL = 1'b1; ADDR1MUX_SEL = 1'b0; ADDR2MUX_SEL = 2'b01; GateMARMUX = 1'b1; LD_MAR = 1'b1;	S_23
		S_23	SR1MUX_SEL = 1'b0; ALUK = 2'b11; GateALU = 1'b1; LD_MDR = 1'b1;	S_16_1
		S_16_1 S_16_2 S_16_3	Mem_OE = 1'b0; Mem_WE = 1'b1;	S_18

PAUSE	1101	PauseIR1	LD_LED = 1'b1;	PauseIR2
		PauseIR2	-----	S_18

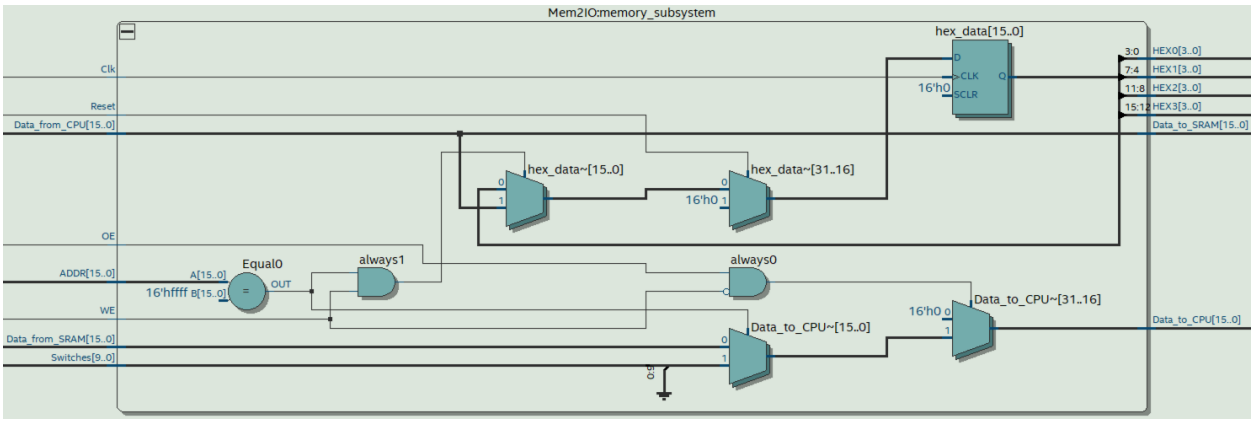
FSM Diagram



Mem2IO High-Level Block Diagram



Mem2IO Block Diagram



Module Description

Module: Mem2IO.sv

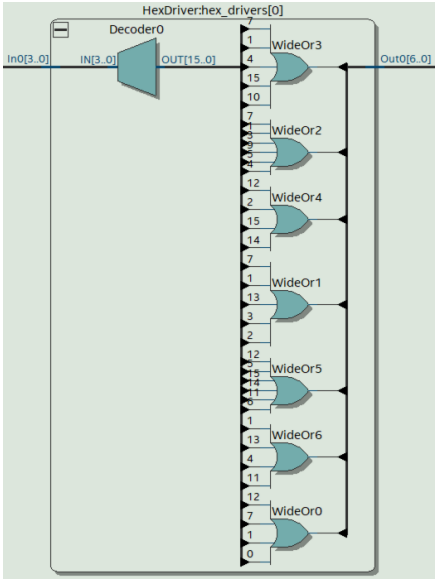
Input: Clk, Reset, [15:0] ADDR, OE, WE, [9:0] Switches, [15:0] Data_from_CPU, [15:0] Data_from_SRAM

Output: [15:0] Data_to_CPU, [15:0] Data_to_SRAM, [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, [6:0] HEX3

Description: This component manages all the I/O with the DE10-Lite physical I/O devices: switches and 7-segment displays. When WE is low and OE is high, Data_to_CPU = Data_from_SRAM. When WE is low, OE is high, and [15:0] ADDR = xFFFF, Data_to_CPU = 16-bit sign-extended [9:0] Switches. When WE is high and [15:0] ADDR = xFFFF, hex_data = Data_from_CPU.

Purpose: This module is used as interface between the CPU and the SRAM (physical on-chip memory). It is not only used to display the memory content through 7-segment displays, but also used to handle 2 special cases which are reading from address xFFFF and storing into address xFFFF.

HexDriver Block Diagram



Module Description

Module: HexDriver.sv

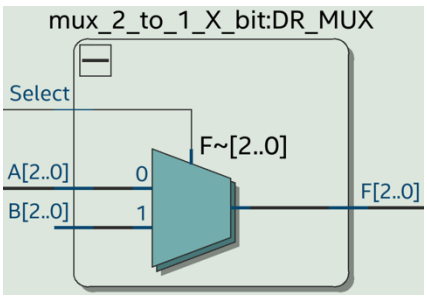
Input: [3:0] In0

Output: [6:0] Out0

Description: This component contains a decoder which decodes the each 4-bit binary value to its corresponding 7-bit binary control value of LED segments. The input [3:0] In0 is translate to 7-bit control value of LED segments [6:0] Out0 which is capable of displaying the hexadecimal letter (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F) on the HEX units.

Purpose: This module is used as 7-Segment Display which can show the 4-bit hexadecimal (2-bit hexadecimal value representing 8-bit binary value in register A and 2-it hexadecimal value representing 8-bit binary value in register B) values to the user on the 4 HEX units.

X-bit 2-to-1 Multiplexer Block Diagram



Module Description

Module: X-bit_2-to-1_Multiplexer.sv (Default width = 16)

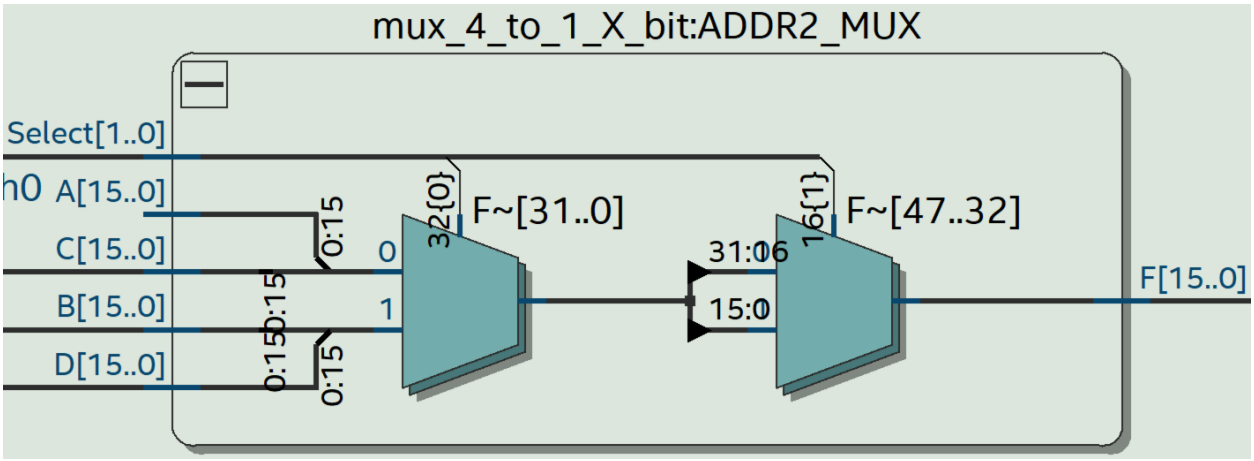
Input: [width-1:0] A, [width-1:0] B, Select

Output: [width-1:0] F

Description: This component is a X-bit 2-to-1 Multiplexer which has dynamic width. When Select = 1'b0, F = A. When Select = 1'b1, F = B.

Purpose: This module is used to build the DRMUX (width = 3), SR1MUX (width = 3), SR2MUX, ADDR1MUX, MDR_MUX.

X-bit 4-to-1 Multiplexer Block Diagram



Module Description

Module: X-bit_4-to-1_Multiplexer.sv (Default width = 16)

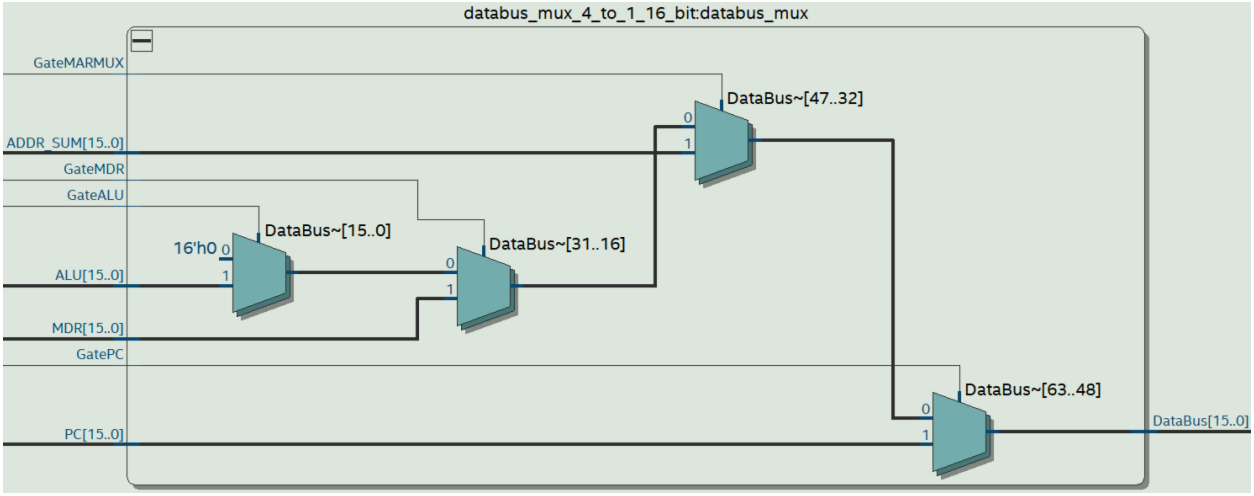
Input: [width-1:0] A, [width-1:0] B, [1:0] Select

Output: [width-1:0] F

Description: This component is a X-bit 4-to-1 Multiplexer which has dynamic width. When Select = 1'b00, F is equal to A. When Select = 1'b01, F = B. When Select = 1'b10, F = C. When Select = 1'b11, F = D.

Purpose: This module is used to build the PCMUX and ADDR2MUX.

16-bit 4-to-1 DataBus Multiplexer Block Diagram



Module Description

Module: 16-bit_4-to-1_DataBus_Multiplexer.sv

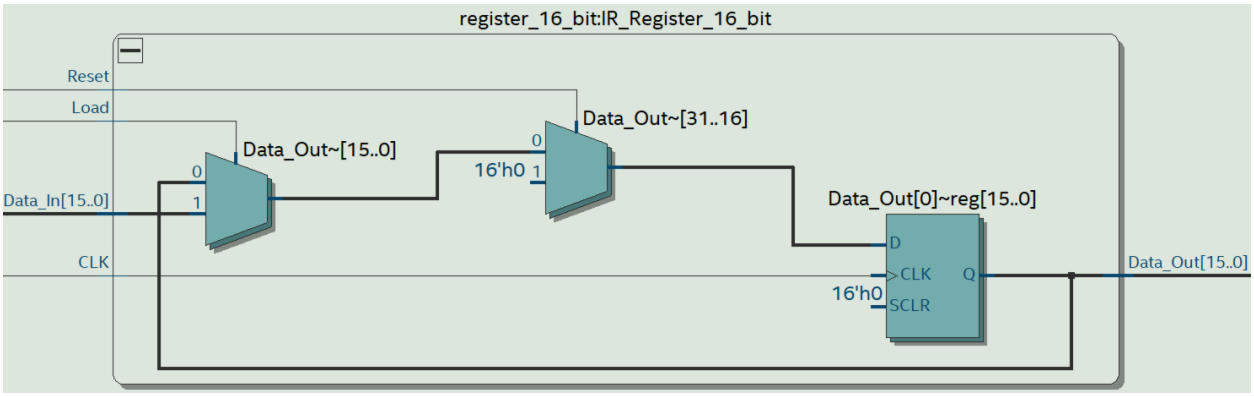
Input: GatePC, GateMARMUX, GateMDR, GateALU, [15:0] PC, [15:0] ADDR_SUM, [15:0] MDR, [15:0] ALU

Output: [15:0] DataBus

Description: This component is a special 16-bit 4-to-1 Multiplexer for DataBus. When GatePC = 1'b1, DataBus = PC. When GateMARMUX = 1'b1, DataBus = MAR. When GateMDR = 1'b1, DataBus = MDR. When GateALU = 1'b1, DataBus = ALU. Else, DataBus = 16'h0.

Purpose: This module is used as the 16-bit 4-to-1 Multiplexer for DataBus since FPGA does not support internal tristate buffers.

16-bit Register Block Diagram



Module Description

Module: 16-bit_Register.sv

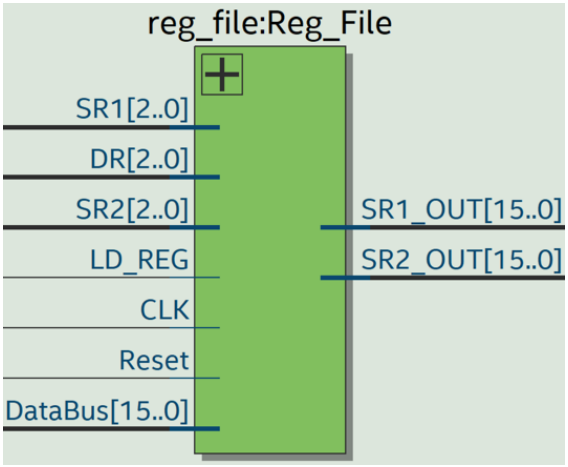
Input: CLK, Reset, Load, Shift_Enable, Shift_In, [15:0] Data_In

Output: Shift_Out, [15:0] Data_Out

Description: This is a positive-edge triggered 15-bit register with synchronous reset and synchronous load. When Reset is high, all the registers are cleared (store the binary value 0) on the positive edge of CLK. When Load is high, data is loaded from [15:0] Data_In into the registers on the positive edge of CLK. Data_Out is the 16-bit output of all the register units (register 0 – register 15).

Purpose: This module is used to build PC register, MAR (Memory Address Register), MDR (Memory Data Register), and IR (Instruction Register).

Reg_File High-Level Block Diagram



Module Description

Module: reg_file.sv

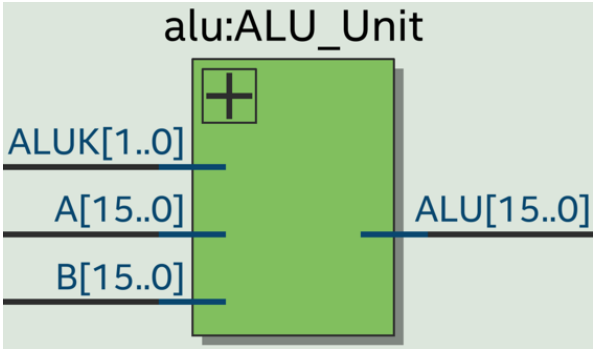
Input: CLK, Reset, LD_REG, [2:0] DR, [2:0] SR1, [2:0] SR2, [15:0] DataBus

Output: [15:0] SR1_OUT, [15:0] SR2_OUT

Description: This component contains eight positive-edge triggered 16-bit registers (R0-R7) with synchronous reset and synchronous load. When Reset is high, all the registers are cleared (store the binary value 0) on the positive edge of CLK. When LD_REG is high, data is loaded from DataBus into 1 of 8 registers specified by DR on the positive edge of CLK. SR1_OUT is the 16-bit output of the register specified by SR1. SR2_OUT is the 16-bit output of the register specified by SR2.

Purpose: This module is high-level block of eight 16-bit registers which are used to either permanently store data or temporarily store data for further operation in SLC-3 processor.

ALU High-Level Block Diagram



Module Description

Module: ALU.sv

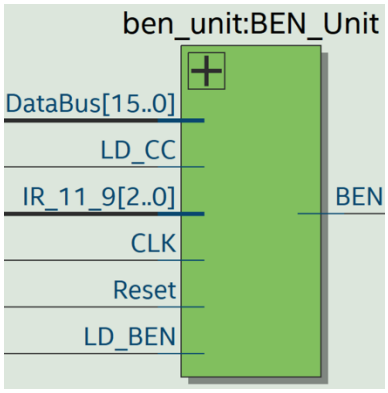
Input: [15:0] A, [15:0] B, [1:0] ALUK

Output: [15:0] ALU

Description: This component is the arithmetic logic unit which is capable of 1 of the 4 operations based on the corresponding [1:0] ALUK select signal: A+B (ALUK = 2'b00), A&B (ALUK = 2'b01), ~A (ALUK = 2'b10), A (ALUK = 2'b11)

Purpose: This module is used as the arithmetic logic unit (ALU) of the SLC-3 processor.

BEN_Unit High-Level Block Diagram



Module Description

Module: BEN_Unit.sv

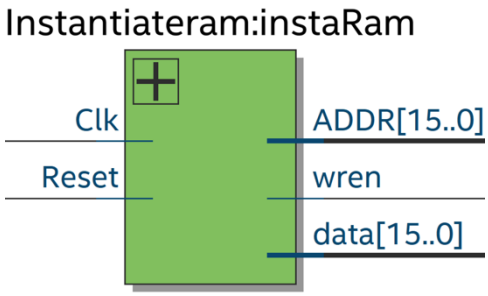
Input: CLK, Reset, LD_BEN, LD_CC, [15:0] DataBus, [2:0] IR_11_9

Output: BEN

Description: This component contains four 1-bit registers for storing the N, Z, P, and BEN values. When [15:0] DataBus is a positive number in 2's complement representation, only P = 1'b1. When [15:0] DataBus is a negative number in 2's complement representation, only N = 1'b1. When [15:0] DataBus = 16'h0, only Z = 1'b1. When LD_CC is high, the three 1-bit registers (N,Z,P) load in the latest value of N, Z, and P. BEN is equal to 1'b1 only if one of the nzp bits IR_11_9 match up with the current NZP values stored in NZP registers. When LD_BEN is high, the 1-bit register BEN loads in the latest BEN value.

Purpose: This module is used to store the status (nzp) of the SLC-3 processor in order to check whether a BR instruction needs to be performed. When SLC-3 processor needs to execute a BR instruction, the module sends out the correct BEN value to the ISDU to ensure the FSM changes to the correct next state.

Instantiateram High-Level Block Diagram



Module Description

Module: Instantiateram.sv

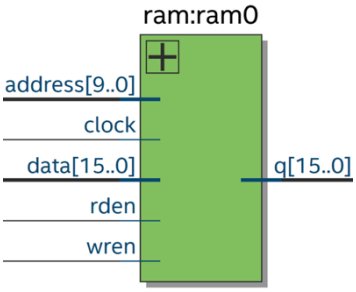
Input: Clk, Reset,

Output: [15:0] ADDR, [15:0] data, wren

Description: This component is used to generate on-chip memory from address 16'd0 to 16'd155 with different content from [15:0] data.

Purpose: This module is used to instantiate on-chip memory.

ram High-Level Block Diagram



Module Description

Module: ram.v

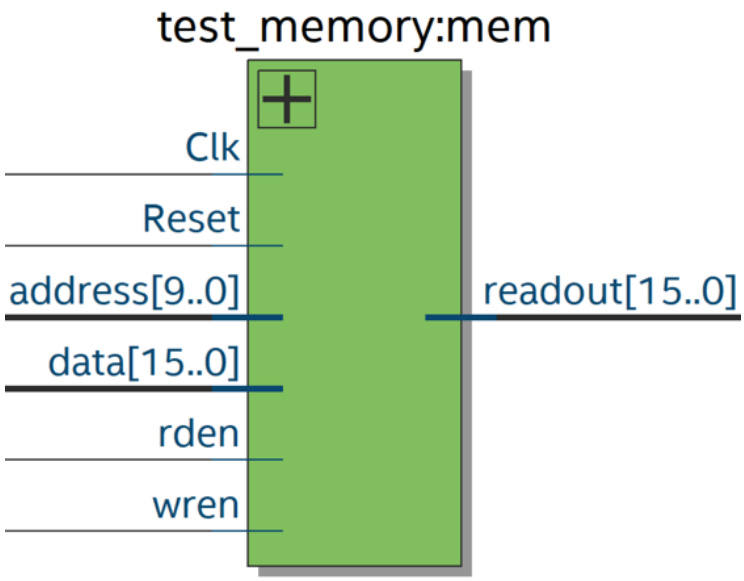
Input: clock, [9:0] address, [15:0] data, rden, wren

Output: [15:0] q

Description: This component is a memory module which has all basic memory inputs and outputs. When rden is high, the memory can only be read and q = M[address]. When wren is high, the memory can only be written and M[address] = data.

Purpose: This module is used as the physical on-chip memory generated by Quartus.

test memory High-Level Block Diagram



Module Description

Module: test_memory.sv

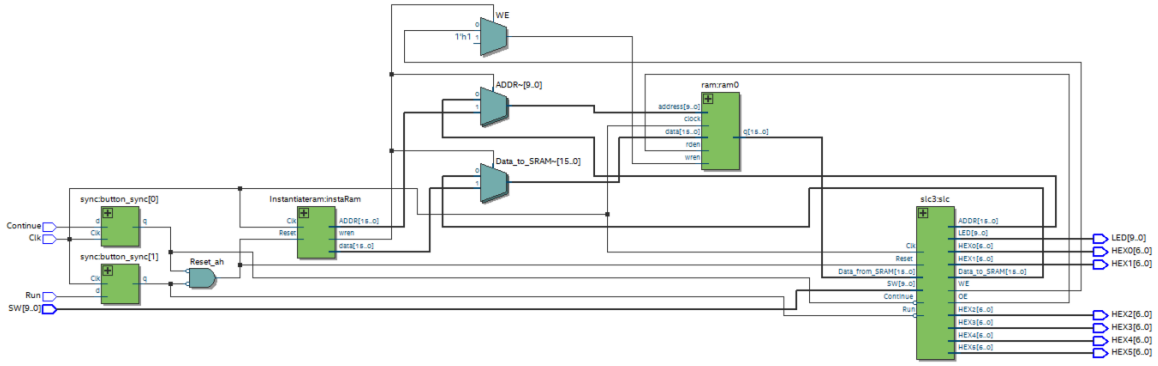
Input: Clk, Reset, [15:0] data, [9:0] address, rden, wren

Output: [15:0] readout

Description: This component is a memory module which has all basic memory inputs and outputs. When rden is high, the memory can only be read and q = M[address]. When wren is high, the memory can only be written and M[address] = data.

Purpose: This module is used for simulation only which performs a similar behavior as the SRAM on the MAX10 board.

slc3_sramtop Block Diagram



Module Description

Module: slc3_sramtop.sv

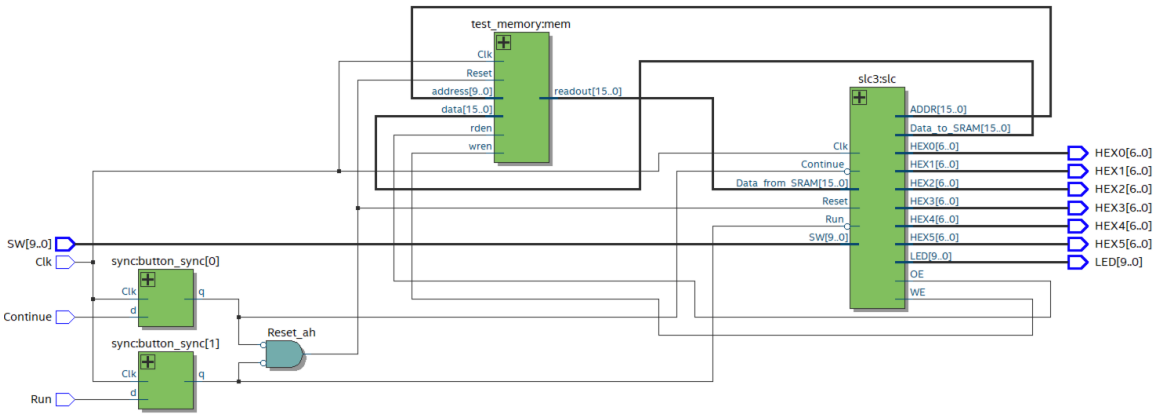
Input: Clk, Run, Continue, [9:0] SW,

Output: [9:0] LED, [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, [6:0] HEX3, [6:0] HEX4, [6:0] HEX5

Description: This component contains the synchronizers, physical on-chip memory, and the CPU.

Purpose: This module is used as the high-level block of the entire SLC-3 processor with the physical on-chip memory. It is the top-level module for testing SLC-3 processor on the MAX10 board.

slc3_testtop Block Diagram



Module Description

Module: datapath.sv

Input: Clk, Run, Continue, [9:0] SW,

Output: [9:0] LED, [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, [6:0] HEX3, [6:0] HEX4, [6:0] HEX5

Description: This component contains the synchronizers, physical on-chip memory, and the CPU.

Purpose: This module is used as the high-level block of the entire SLC-3 processor with test memory. It is the top-level module for testing SLC-3 processor in testbench.

SLC3_2

Module: SLC3_2.sv

Description: This module is used as library reference which contains translation between lc3 assembly language to binary values and all the lc3 operations in the form of functions.

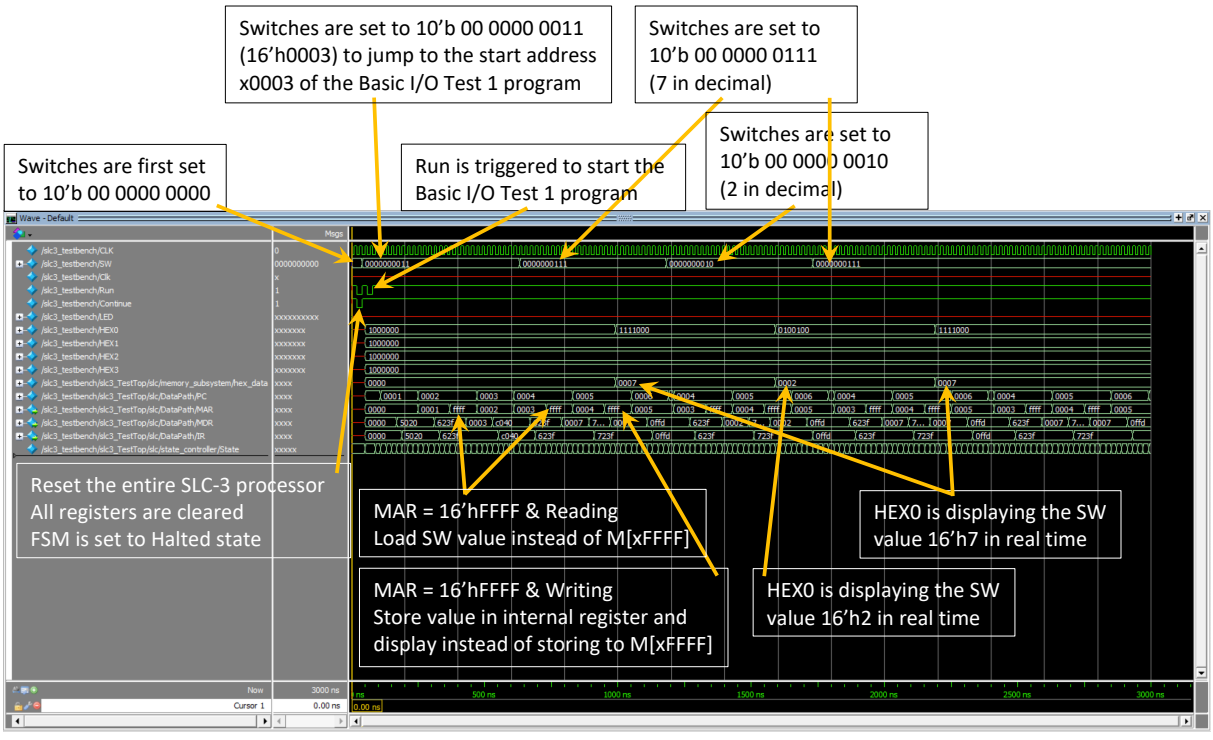
memory_contents

Module: memory_contents.sv

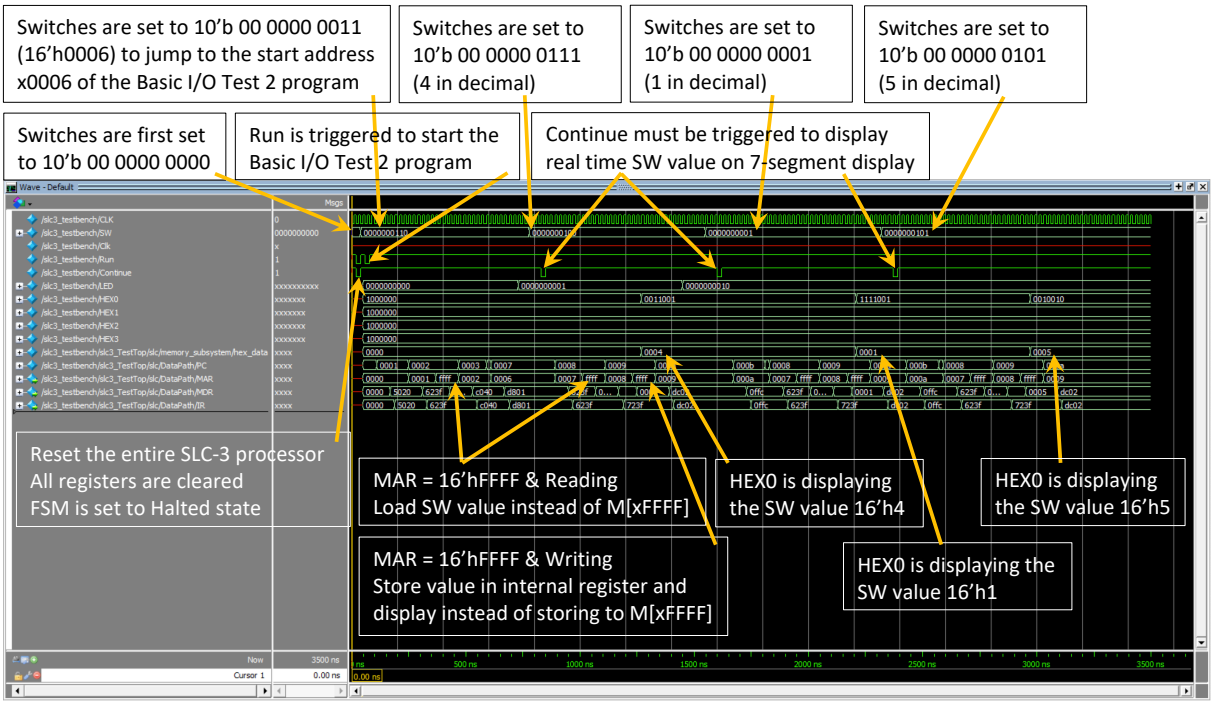
Description: This module contains the memory contents in the test_memory.sv.

Simulations of SLC-3 Instructions

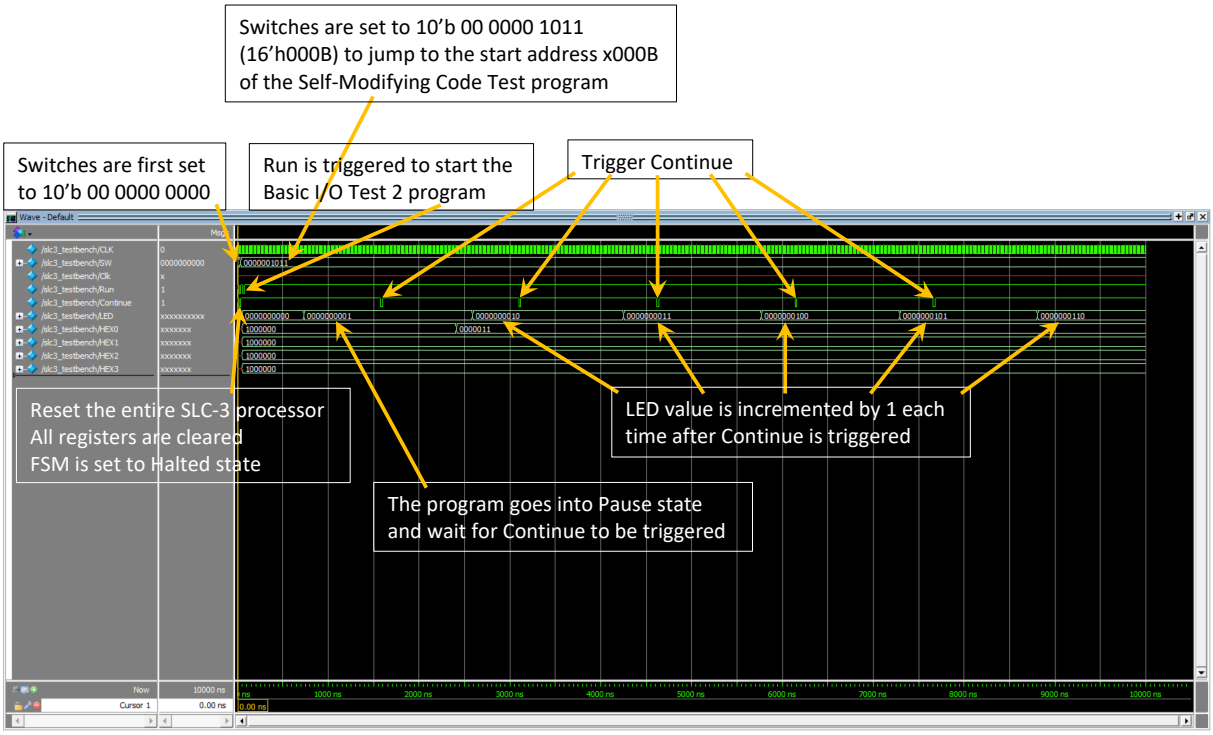
Basic I/O Test 1



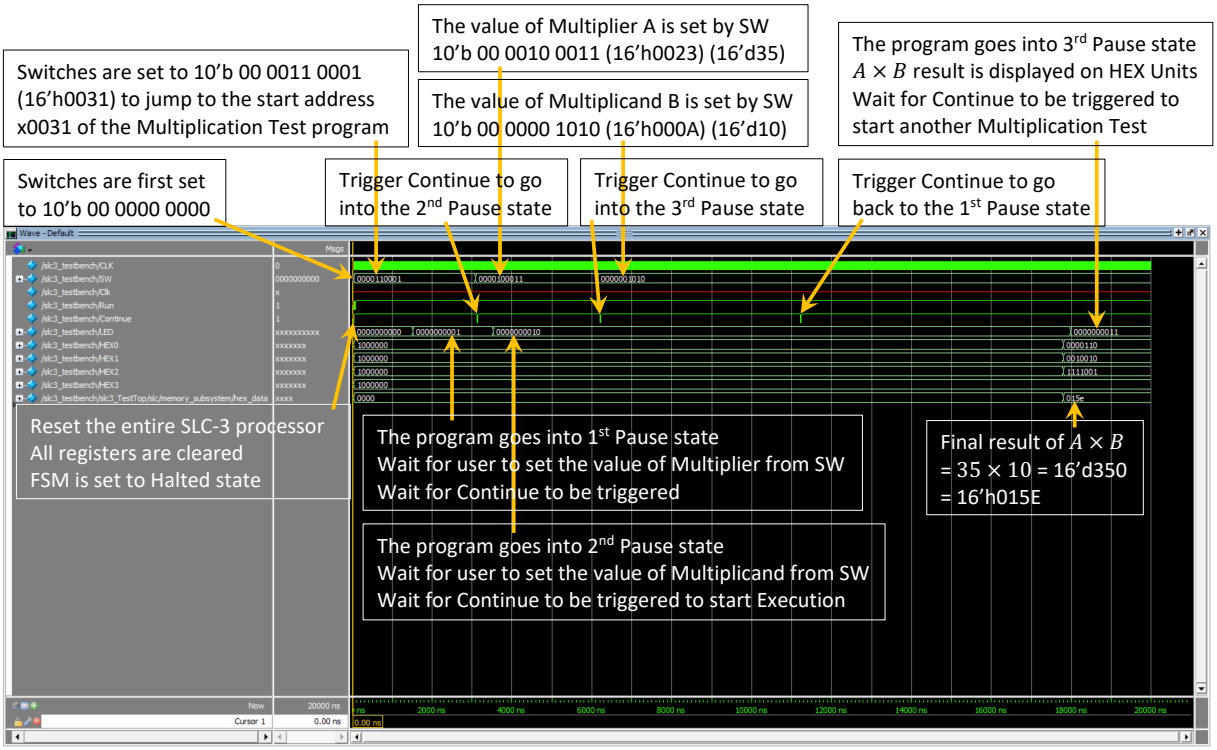
Basic I/O Test 2



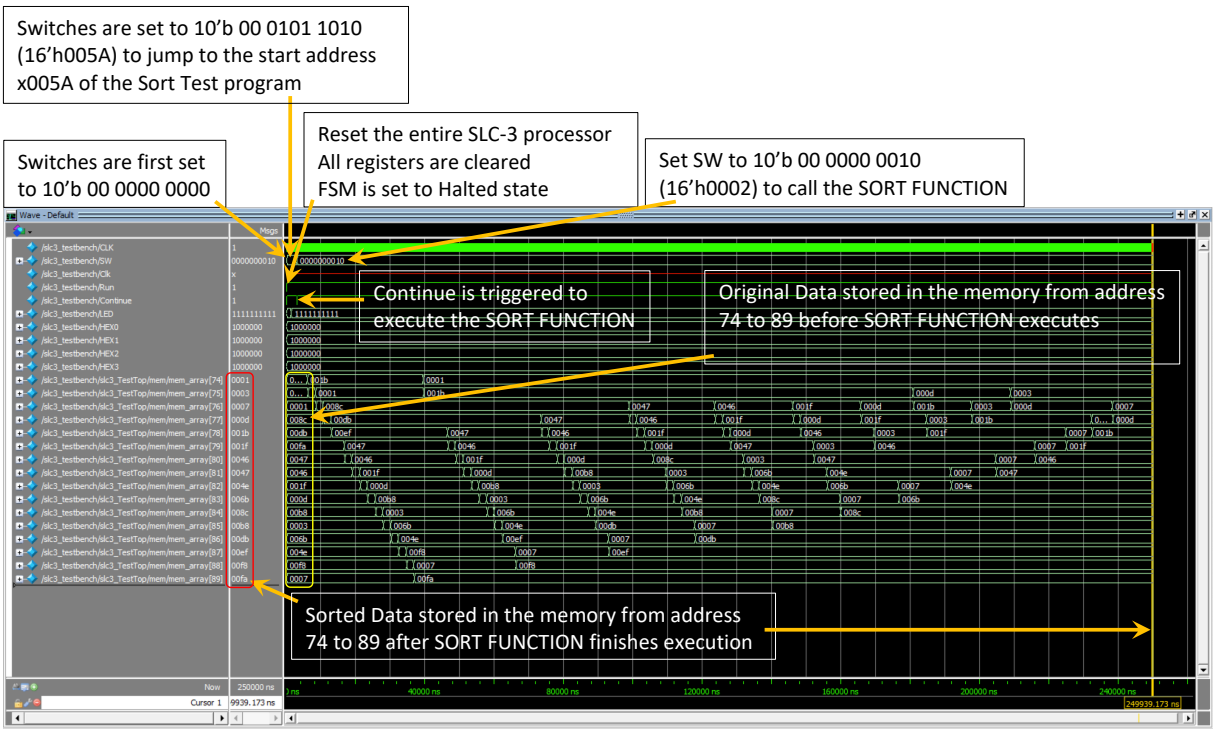
Self-Modifying Code Test



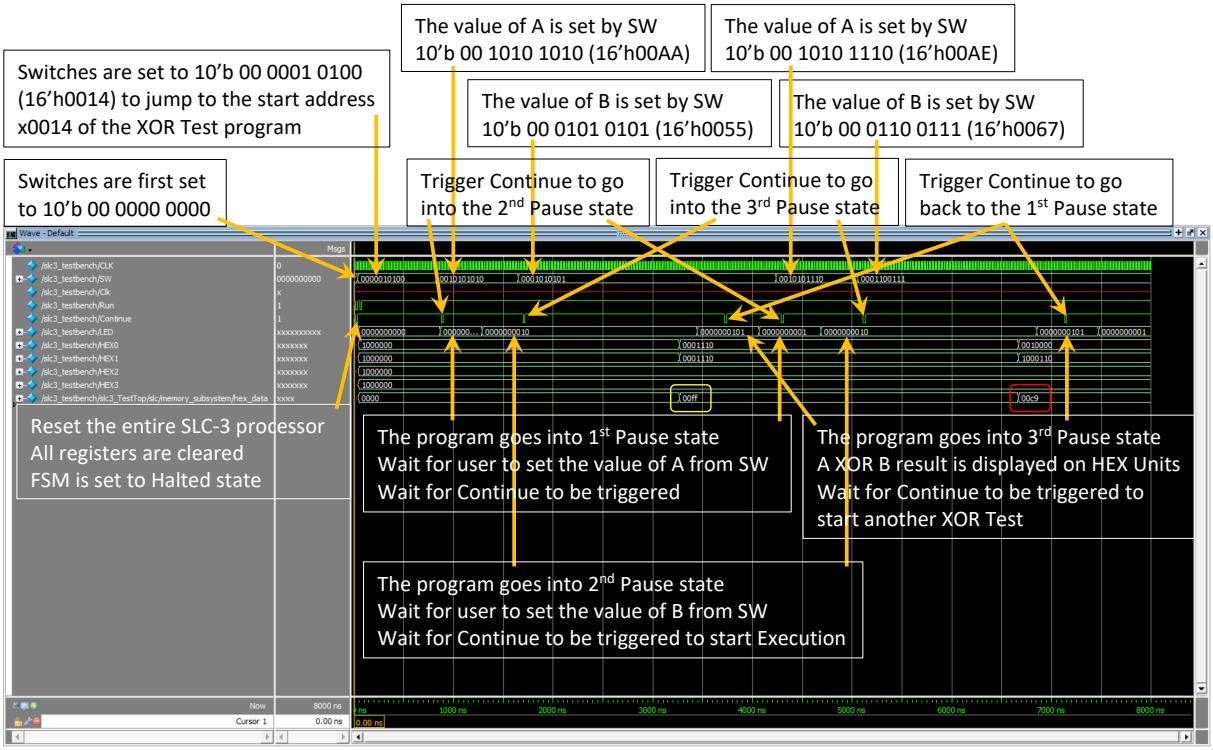
Multiplication Test



Sort Test



XOR Test



Inputs	Values of Inputs in Binary	Values of Inputs in Hexadecimal
Input A1	16'b 0000 0000 1010 1010	16'h 00AA
Input B1	16'b 0000 0000 0101 0101	16'h 0055
Input A2	16'b 0000 0000 1010 1110	16'h 00AE
Input B2	16'b 0000 0000 0110 0111	16'h 0067

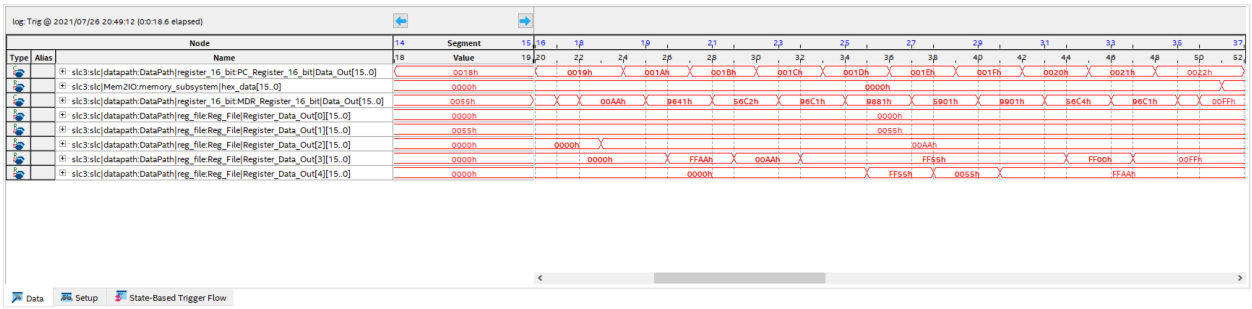
Operation	Result of A XOR B in Binary	Result of A XOR B in Hexadecimal
A1 XOR B1	16'b 0000 0000 1111 1111	16'h 00FF (Yellow Circle in Figure)
A2 XOR B2	16'b 0000 0000 1100 1001	16'h 00C9 (Red Circle in Figure)

Bugs encountered and corrective measures taken

This lab mainly requires the implementation of datapath and ISDU. After finishing the implementation of both modules, the SLC-3 processor project compiles without giving out errors. However, the Basic I/O Test 1 does not work properly after programming the slc3_sramtop on the DE10-Lite board since the Hex Units displays x0000 instead of the SW value in real time. Then, testbench is used to debug for the datapath and ISDU modules after changing the top-level to slc3_testtop. After adding PC, MAR, MDR, IR, and state into the waveform, the value of PC seems to be wrong which makes the value of IR inconsistent with the memory contents. Therefore, the output of PCMUX is added to the waveform, and it turns out that the PCMUX_SEL signal is choosing the wrong input. After changing the value of PCMUX_SEL in ISDU, all the 6 test programs are able to function correctly on the DE10-Lite board.

SignalTap Traces

XOR Test



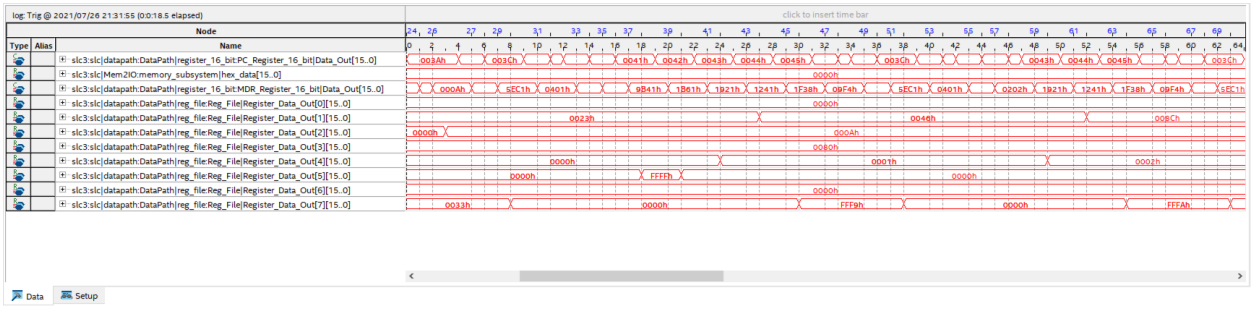
*** Performing the same XOR Test as the ModelSim: x0055 XOR x00AA = x00FF ***

The core algorithm of XOR starts from address x0019 to address x0020. Therefore, there are 8 instructions to execute. The moment when the XOR result is ready in R3 is at 47ns. It takes $47ns - 20ns = 27ns$ to finish the core algorithm of XOR.

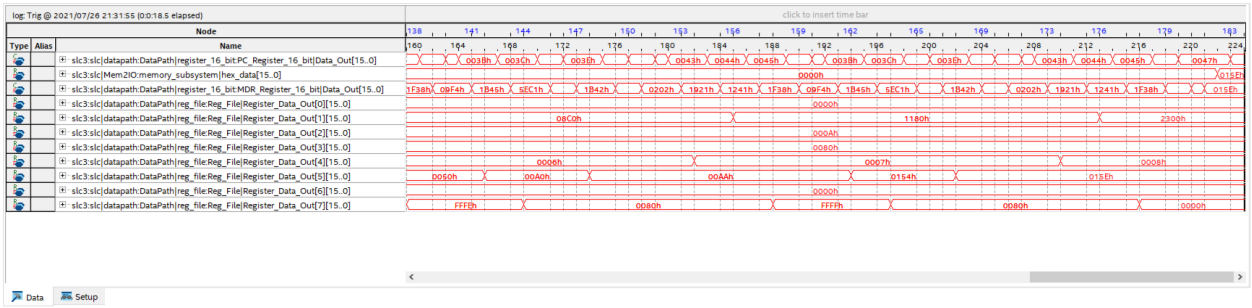
The CPU is capable of $\frac{8\ instructions}{27\ cycles} \times 50M\ \frac{cycles}{s} \approx 14.815\ MIPS$ on XOR Test.

Multiplication Test

Start of Core Multiplication algorithm at address x003A



End of Multiplication algorithm



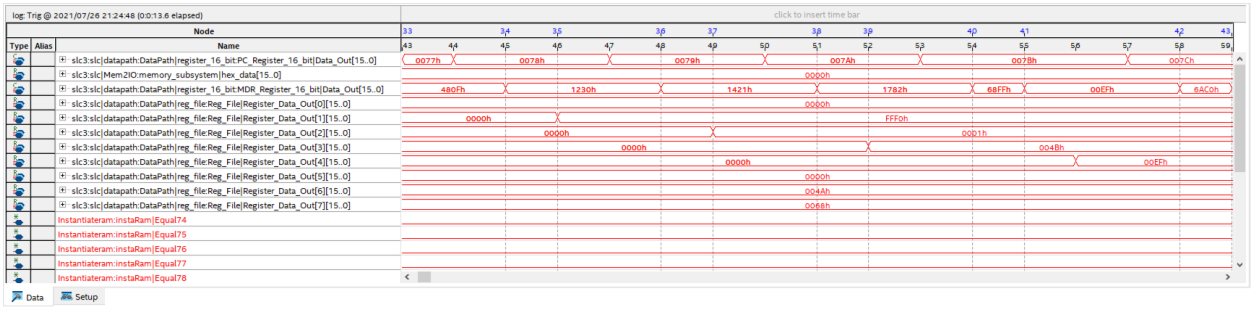
*** Performing the same Multiplication Test as the ModelSim: x0023 (35 in decimal) × x000A (10 in decimal) = x015E (350 in decimal) ***

The core algorithm of Multiplication starts from address x003A, and there are 90 instructions to execute. The moment when the result of Multiplication is ready in R5 is at 202ns. It takes $202ns - 0ns = 202ns$ to finish the core algorithm of Multiplication.

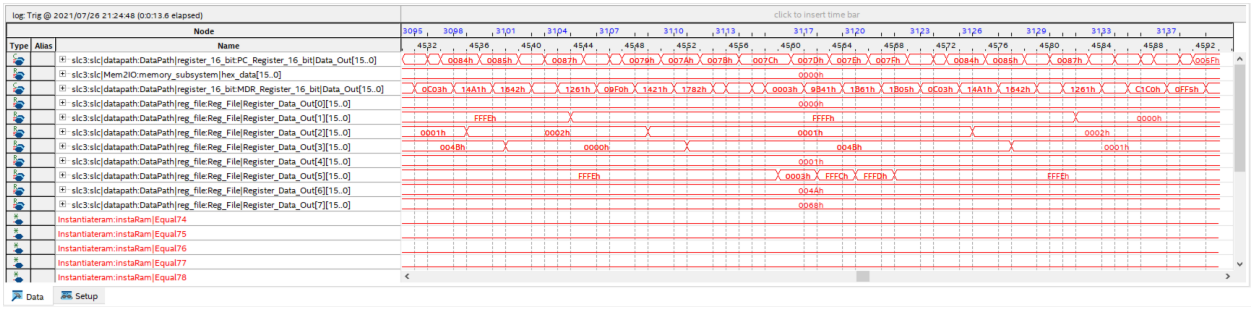
The CPU is capable of $\frac{90\ instructions}{202\ cycles} \times 50M\ \frac{cycles}{s} \approx 22.2772\ MIPS$ on Multiplication Test.

Sort Test

Start of the Sort Function at address x0077



End of the Sort Function



The core algorithm of Sort Function starts from address x0077, and there are 1600 instructions to execute. The moment when the result of Sort Function is ready is at 4592ns. It takes $4592ns - 43ns = 4549ns$ to finish the core algorithm of Sort Function.

The CPU is capable of $\frac{1600\ instructions}{4549\ cycles} \times 50M\ \frac{cycles}{s} \approx 17.5863\ MIPS$ on Sort Function Test.

$(14.815\ MIPS + 22.2772\ MIPS + 17.5863\ MIPS) \div 3 \approx 18.2262\ MIPS$

Therefore, the average SLC-3 MIPS is $18.2262\ MIPS$.

Post-Lab

Use lc3_sramtop.sv as top-level and Compile

	SLC-3 Processor
LUT	894
DSP	None
Memory (BRAM)	18432
Flip-Flop	268
Frqeuency	68.94MHz
Static Power	90.01mW
Dynamic Power	8.19mW
Total Power	112.77mW

Use lc3_testtop as top-level and Compile

	SLC-3 Processor
LUT	572
DSP	None
Memory (BRAM)	0
Flip-Flop	251
Frqeuency	66.8MHz
Static Power	90.00mW
Dynamic Power	6.50mW
Total Power	111.1mW

Both BR and JMP are able to perform a change of PC value. However, BR needs to meet a specific condition in order to modify the value of PC whereas JMP does not need to. The 11^{th} to 9^{th} bit of a BR instruction indicates the nzp value the user want to check. The datapath also has three 1-bit register: N,Z,P which stores the whether the result of last operation is a positive, zero, or negative number (setCC needed). If one of the nzp value in the BR instruction meets the values stored inside the NZP registers in datapath (both nN or zZ or pP have to be 1 at the same time), a BR instruction is performed, and PC value will be changed by adding the 9-bit PC offset to the current PC value. The 9-bit PC offset is a binary number in 2's complement representation, so the PC value may increase or decrease. If none of the nzp value in the BR instruction meets the values stored inside the NZP registers, the BR instruction is not performed, and the current PC value is maintained. On the other hand, the JMP instruction directly loads the value stored inside Base Register into PC which performs an unconditional change to the PC value.

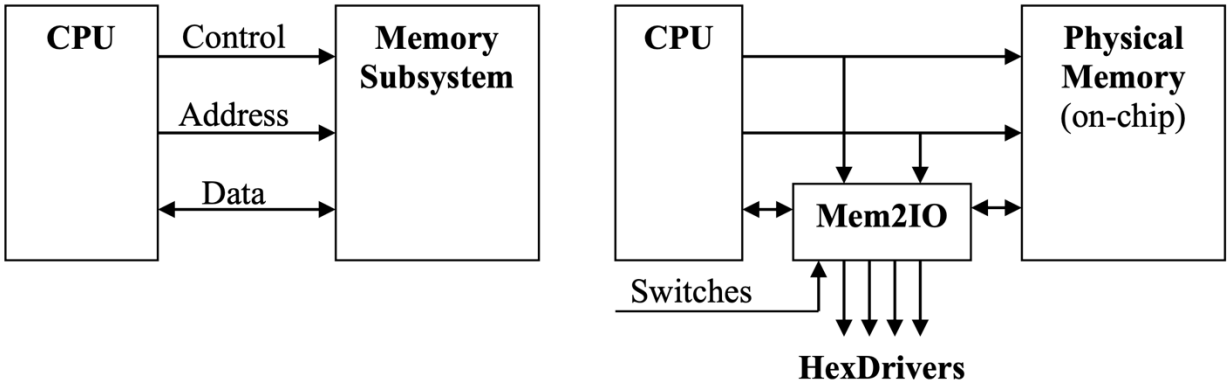
If user wants to go to a specific address, JMP is better than BR because the user does not need to calculate the PC offset by themselves. If the user wants to perform a for loop, BR is better than JMP because the user can decide how many times to execute the loop.

In some cases, BR and JMP are similar instruction when all nzp inside the BR instruction are 1. Since at least one of the NZP registers in the datapath is going to be 1, the BR instruction will 100% execute. Therefore, the BR instruction changes from a conditional instruction to an unconditional instruction which is very similar to JMP.

The MEM2IO is used to manage all I/O with the DE10-Lite physical I/O devices: switches and 7-segment displays. Originally, the CPU should have its Control, Address, and Data (for data transmission) wires directly connected from the CPU to the Physical Memory (on-chip) as shown in the left figure. However, the Mem2IO module is added between the 2 data transmission wires (shown in the right figure): Data from CPU to SRAM and Data from SRAM to CPU in order to allow the user to interact with the SLC-3 processor, such as asking for inputs from the user and displaying memory contents on Hex Displays. The MEM2IO allows the interaction between user and SLC-3 processor through 2 ways. First, when the MEM2IO detects load from address 0xFFFF, it interrupts this action and loads the value from the SW instead of M[xFFFF] from SRAM. Second, when the MEM2IO detects store to address 0xFFFF, it interrupts this action and store the value in internal register and display it instead of storing into SRAM.

For example, the first 3 instructions inside the memory are CLR R0, LDR R1 R0 insw(xFFFF), and JMP R1. The second instruction is loading from the memory content at address xFFFF which is interrupted by MEM2IO and changed to load SW value. Therefore, the third instruction can let the program jump to the address which is assigned by the user from the switches.

Another example is the fifth instruction inside the memory, which is STR R1, R0, outHEX(xFFFF). This instruction is storing into memory at address xFFFF which is interrupted by MEM2IO and changed to store into internal register and display it through Hex Units. Therefore, the user can see the Hex Units are displaying the SW value assigned by the user in real time.



The R signal in Patt and Patel enables the memory read or write to execute correctly. Since the memory knows that it will need several clock cycles (assume 5 clock cycles) to complete the read or write process, the memory will send out the R=1 signal throughout the fifth clock cycle to make sure the read from or write to the memory is executed successfully. In SLC-3 processor, it is known that the memory read or write requires at least 3 clock cycles, so 2 extra read states (MDR <- M[MAR]) are added for state 33 and state 25, and 2 extra write states (M[MAR] <- MDR) are added for state 16 to compensate the lack of the R signal of the memory. Therefore, adding extra states have the same functionality as the 'R' signal because they both wait for several clock cycles to make sure the read from and write to processes are executed successfully. They both avoid the memory and the MDR gets the wrong data. Specifically, adding extra states will avoid two bad cases from happening. The first one is that the memory is not ready for writing, and the MDR value changes which will make the memory get the wrong data. The second one is that the memory is not ready for reading, and the MDR gets the wrong value from other places.

Conclusion

The functionality of the design for this lab is a SLC-3 processor which is able to read from the memory/registers, write to the memory/registers, change PC (program counter) value, handle different status (nzp), do addition, and perform simple logic operations: AND, NOT. The memory component and the interface between the memory and the CPU are given which significantly decreases the difficulty of SLC-3's implementation. Since there are many 2-to-1 multiplexers with different input width inside the datapath, having a 2-to-1 multiplexer with dynamic width avoids redundant module creation. In addition, the usage of unpacked array of packed logic makes the design of Register File simpler because it eliminates the necessity of creating eight identical 16-bit registers inside the Register File. Therefore, using behavioral HDL decreases the number of codes needed for creating the Register File compared to using the structural HDL. The IR5 input of the ISDU can be replaced by a wire inside the datapath which directly connects the IR[5] and SR2MUX in order to simplify the design of ISDU. The SLC-3 datapath diagram and SLC-3 state diagram in the lab manual are really helpful for creating the datapath and ISDU modules. Moreover, the MEM2IO question makes us understand how it allows the user to interact with the SLC-3 processor, such as asking for inputs from the user and displaying memory contents on Hex Displays.