# ECE 385

Fall 2020

Experiment # 6

**Simple Computer SLC-3.2 in SystemVerilog**

Lian Xinyu & Liu Tianyu

TA: Li Shuren

**Introduction**

In this lab, we make a simple computer--SLC3, which can perform some basic functions for PC (personal computer), such as calculation, sort, etc. To perform such functions, this simple computer will receive the instructions and fetch them, then decode them and execute the instructions. To successfully build the LC3 computer, using CAD, we designed all the components needed for the computer and completed modifications to the state machine. After two weeks of testing and demonstration of the operation, we consider the lab a success. The software we use is Quartus v18.1.

**Written Description and Diagrams of SLC-3**

1. **Summary**

According to our design, the SLC-3 can perform these following functions: ADD/ADDi/AND/ANDi/NOT; BR/JMP/JSR; LDR/STR;/PAUSE. All of the functions will follow this cycle: fetch, decode and execute.

图示

描述已自动生成

1. **Description**

The Fetch-Decode-Execute cycle is controlled by the control unit of SLC3 (ISDU.sv), which is known as the state machines for this processor. Referring to the content of control unit logic, we firstly need to initialize all the control signals. If the run signal is detected, the state machine will go state S\_18, which can load the value of PC into MAR and increase the value of PC. In addition, S\_18 will then go the state S\_33, which performs the function that loading the data in MAR(address) to MDR. If signal R is true, the state will go to S\_35 and the value of MDR will be loaded into IR. Above all, S\_18, S\_33 and S\_35 will combine as FETCH part.

The next component is DECODE part, which is known as state S\_32. This state firstly assigns a value to BEN by comparing the values of IR[11:9] and NZP to determine whether a BR instruction should be executed. The state then compares IR[15:11] to determine which instruction to execute in the following steps (e.g. ADD, JMP).

Because different instructions correspond to slightly different states, so SLC3 will work in accordance with the instruction (IR[15:11]) state transfer mode, after the work of this instruction is completed, the state will jump back to S\_18 (for example, for the instruction ADD, the state transfer path is S\_32->S\_1->S\_18).

In total, the process or state machines are corresponding to the FETCH-DECODE-EXECUTE cycle.

1. **d. Diagram**

**Overall**

图片包含 图示

描述已自动生成

Left part

图示, 示意图

描述已自动生成

Middle part

图片包含 图示

描述已自动生成

Right part

图示, 示意图

描述已自动生成

1. **Written Description of all .sv modules**
2. **Module: lab6\_toplevel.sv**

**Inputs:**

**Logic [15:0] S,**

**Logic Clk, Reset, Run, Continue,**

**Outputs:**

**Logic [11:0] LED,**

**Logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7,**

**Logic CE, UB, LB, OE, WE,**

**Logic [19:0] ADDR,**

**Inout:**

**Wire [15:0] Data**

**Description:** The lab6\_toplevel can feed the correct signal into our slc3 processor and the supplied test\_memory module, and produce the output of the corresponding modules.

**Propose:** Generating the output of slc3.sv and test\_memory.sv

1. **Module: ISDU.sv**

**Inputs:**

**Logic Clk, Reset, Run, Continue,**

**Logic [3:0] Opcode,**

**Logic IR\_5,**

**Logic IR\_11,**

**Logic BEN**

**Outputs:**

**Logic LD\_MAR, LD\_MDR, LD\_IR, LD\_BEN, LD\_CC, LD\_REG, LD\_PC, LD\_LED, GatePC, GateMDR, GateALU, GateMARMUX,**

**Logic [1:0] PCMUX,**

**Logic DRMUX, SR1MUX, SR2MUX, ADDR1MUX,**

**Logic [1:0] ADDR2MUX, ALUK,**

**Logic Mem\_CE, Mem\_UB, Mem\_LB, Mem\_OE, Mem\_WE**

**Description:** This module is the state machine for controlling the operations of slc3 processor. It will deal with the instructions that receiving from the input and control the processor to finish the corresponding tasks.

**Propose:** Control the operations of our slc3 processor.

1. **Module: HexDriver.sv**

**Inputs:**

**Logic [3:0] In0**

**Outputs:**

**Logic [6:0] Out0**

**Description:** This module will transfer the input signal to 7 bits form which can be read using LEDS.

**Propose:** In week1, it is used to show IR using LEDS; In week2, it is used to be mounted to Mem2IO.

1. **Module: tristate.sv (N=16)**

**Inputs:**

**Logic Clk,**

**Logic tristate\_output\_enable,**

**Logic [N-1:0] Data\_write,**

**Outputs:**

**Logic [N-1:0] Data\_read,**

**Inout:**

**Wire [N-1:0] Data**

**Description:** This module gives the design for a tristate (also known as tristate buffer). Here is a truth table for 1 bit tristate (EN: tristate\_output\_enable, IN:Data\_write, OUT: Data\_read ,the output will go to the containers of Data.)

|  |  |  |
| --- | --- | --- |
| **EN** | **IN** | **OUT** |
| 0 | 0 | Z |
| 0 | 1 | Z |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Z means highs impedence.

**Propose:** In our slc3 processor, the tristate buffer will serve as the interface between Mem2IO and SRAM.

1. **Module: test\_memory.sv**

**Inputs:**

**Logic Clk, Reset,**

**Logic [15:0] I\_O,**

**Logic [19:0] A,**

**Logic A, CE, UB, LB, OE, WE.**

**Description:** This module is used to simulate the content and performance of the SRAM IC on the DE2 board, which means it will not participate in our testing using the DE2 board. It can work as a real SRAM.

**Propose:** Simulate the SRAM IC on the DE2 board.

1. **Module: SLC3\_2.sv**

**Description:** This is a package file which the CPU in SLC3 can use to understand the meaning of 16 bit instructions. This library reference can be used for translating all of the instructions involved in the the SLC3 processor.

**Propose:** Tell the meaning of 16-bit input instructions

1. **Module: slc3.sv**

**Inputs:**

**Logic [15:0] S,**

**Logic Clk, Reset, Run, Continue,**

**Outputs:**

**Logic [11:0] LED,**

**Logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7,**

**Logic CE, UB, LB, OE, WE,**

**Logic [19:0] ADDR,**

**Inout:**

**Wire [15:0] Data**

**Description:** This module is the main hardware that contains the CPU part of SLC3, including different registers, MUXs and control unit used in our design. This module can take the instructions out of SRAM part, and pass the output value to the address part and wires of data.

**Propose:** Perform the functions same as a micro CPU.

1. **Module: memory\_content.sv**

**Outputs:**

**Logic [15:0] mem\_array[0:size-1]**

**Description:** This file includes the content for the SRAM, which can be used to simulate the instructions stored in the real LC3 machine. We can also modify the instructions and rearrange the sequence of instructions to test our design.

**Propose:** Simulate the situation in SRAM.

1. **Module: Mem2IO.sv**

**Inputs:**

**Logic [19:0] ADDR,**

**Logic CE, UB, LB, OE, WE,**

**Logic [15:0]** **Switches,**

**Logic [15:0] Data\_from\_CPU, Data\_from\_SRAM,**

**Outputs:**

**Logic [15:0] Data\_to\_CPU, Data\_to\_SRAM,**

**Logic [3:0] HEX0, HEX1, HEX2, HEX3**

**Description:** This is used as a bridge among the CPU, the SRAM and the switches used in the DE2 board for data input. Firstly, it can pass the data from switches to SRAM or CPU; secondly, it can pass the data from CPU to SRAM; in addition, it can also write to LEDs when WE is active and address is xFFFF.

**Propose:** A bridge among CPU, SRAM and switches.

1. **Module: ALU.sv**

**Inputs:**

**Logic [15:0] A,B,**

**Logic [1:0] ALUK**

**Outputs:**

**Logic [15:0] ALU\_OUT**

**Description:** This is the computing unit used by the CPU of slc3 processor. According to the value of ALUK, for the input data A and B, it can perform the following functions as output: A+B, A&B, ~A and A.

**Propose:** A computing module.

1. **Module: MUX.sv (k-to-1 multiplexer)**

**Inputs:**

**Logic [width-1:0] data0,data1,…,data(k)**

**Logic [log(k):0] select (log means log\_2)**

**Outputs:**

**Logic [width-1:0] out**

**Description:** This module includes three kinds of design for different multiplexers:2 to 1, 4 to 1 and 8 to 1. The multiplexer can produce the output corresponding to the value of select signal (width means the length of data).

**Propose:** The design for three kinds of multiplexers.

1. **Module: datapath.sv**

**Inputs:**

**Logic [15:0] PC, ADDER\_OUT, ALU\_OUT, MDR,**

**Logic GatePC, GateMDR, GateALU, GateMARMUX,**

**Outputs:**

**Logic [15:0] BUSRESULT**

**Description:** This module is used to determine the content of BUS. The data which is assigned to the BUS is determined by the following signals: PC\_GATE, MDR\_GATE, ALU\_GATE and MARMUX\_GATE. If one of these signals is 1 and the others are 0, then the module will control slc3 processor to input the corresponding data into the BUS. Otherwise it is going to put X into the BUS.

**Propose:** Control the data input for BUS.

1. **Module: reg\_file.sv**

**Inputs:**

**Logic [15:0] BUSRESULT**

**Logic [2:0] DR, SR1, SR2,**

**Logic LD, Clk, Reset,**

**Outputs:**

**Logic [15:0] SR1\_OUT, SR2\_OUT**

**Description:** This module is the set of registers in slc3 processor, which can be found in the high level schematic in our lecture slides. According to the design of slc3, this module contains 8 registers, R0-R7, and two multiplexers working for selecting the output of SR1 and SR2.

**Propose:** Storage of data during processing instructions

1. **Module: BEN.sv—NZP\_LOGIC**

**Inputs:**

**Logic [15:0] BUSRESULT,**

**Outputs:**

**Logic N, Z, P**

**Description:** This module is to set the value of N, Z and P, which can be determined from whether the value in BUS is negative, positive or zero

**Propose:** Produce correct NZP value used in BR instruction.

1. **Module: BEN.sv—BEN\_LOGIC**

**Inputs:**

**Logic N, Z, P**

**Logic [2:0] c,**

**Outputs:**

**Logic BENRESULT**

**Description:** This module is only related to the BR instructions. If the BR instruction is need to perform, it will pass the control signal to execute this instruction, which means jump to another address.

**Propose:** Generate the control signal for BR instruction.

1. **Module: testbench.sv**

**Description:** There are two kinds of testbench files used in lab6, testbench\_week1 and testbench\_week2, and they have different functions. The former one is used to check the correctness of fetch and decode step, while the later one is used to perform the full instructions.

**Propose:** Test our design

1. **Module: Synchronizers.sv—sync\_r1/r16**

**Inputs:**

**Logic Clk, Reset,**

**Logic [15:0] or Logic d**

**Outputs:**

**Logic[15:0] q or Logic q**

**Description:** These synchronizers are required for bringing asynchronous signals into the FPGA. Performing as a kind of registers, the synchronizer can work with reset to 1 and keep the stability of the processor.

**Propose:** Eliminate the errors carried by asynchronous signals.

**Simulation Results**

**We choose these following instructions to display: ADD, BR and LDR.**

1. **ADDi (opADDi(R2, R2, 1) , in testbench\_week2)**

**电脑萤幕画面

描述已自动生成**

Begin with increasing the value in PC, the slc3 processor will finish the fetch and decode step, then pass the instruction to BEN step(State 32). The instruction will be treat as ADDi(R2, R2, 1), which means add 1 and value in R2 to register R2.

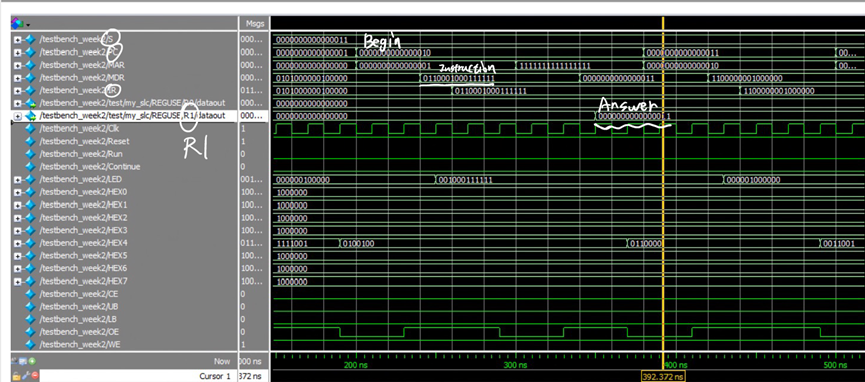
1. **BR (opBR(nzp,-3), in testbench\_week2)**

**图片包含 图形用户界面

描述已自动生成**

Begin with increasing the value in PC, the slc3 processor will finish the fetch and decode step, then pass the instruction to BEN step(State 32). The instruction will be treat as BR(nzp,-3), which means it will jump to the former address, which contains LDR instruction.

1. **LDR (opLDR(R1,R0,inSW), in tesebench\_week2)**

****

Begin with increasing the value in PC, the slc3 processor will finish the fetch and decode step, then pass the instruction to BEN step(State 32). The instruction will be treat as LDR(R1, R0, inSW), which means it will load the value in address: value in R0+inSW, to the register R1.

**Post lab**

1.

|  |  |
| --- | --- |
| **LUT** | 576 |
| **DSP** | No DSP |
| **Memory (BRAM)** | 0 |
| **Flip-Flop** | 288 |
| **Frequency(MHz)** | 67.24 |
| **Static Power(mW)** | 98.65 |
| **Dynamic Power(mW)** | 7.97 |
| **I/O Thermal Power(mW)** | 70.15 |
| **Total Power(mW)** | 176.77 |

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

By analyzing the code content, MEM2IO will be used for controlling the interaction among CPU, SRAM, and I/O operation (switches). The detailed functions that this module will perform depend on the value of control signal, normally they are the value in certain address, WE and OE. When WE is assigned (write able) and OE(output able) is forbidden, if the loaded value in the input address is xFFFF, the value of switches will be passed to CPU, otherwise it will be passed into SRAM. This function can be used to perform jump instructions. When WE is forbidden and the loaded value in the input address is xFFFF, the data from CPU will be loaded to display part (LED). Moreover, it also can reset the display value and pass the data from CPU to SRAM.

**3. What is the difference between BR and JMP instructions?**

Firstly, the have different binary value as instructions. Secondly, BR will be acted as jump instruction only in certain condition, which means the result of comparison between IR[11:9] and NZP. It will only jump when the certain condition requirement is fulfilled. However, the JMP instruction is a jump instruction without condition, the CPU and perform this function whatever it executes JMP. Finally, the BR need jump to certain address based on the offset, while JMP instruction is directed by the given destination address (value in BaseR).

**4. 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**

R means “Ready”, which can make sure the operation that can access the SRAM part will be performed in order or in sequence, according to Appendix C [1]. In our design, we create more states to provide enough time for loading or writing the data in current cycle, which means that the instructions will be always performed in order.

**Conclusion**

1. Thanks to Pro. Li, we learnt a lot in the lectures, and we can finish the design of slc3 processor smoothly. However, we still face some problems in CPU construction and ISDU programming. Firstly, we spent plenty of time in finding all components and determine the sequence of input signals. Secondly, when we designed the state of the ISDU set, it is easy to input the wrong control signal, which led to many problems with the CPU operation. Thirdly, for the using of synchronizers, at first we did not use them. To me surprise, during the test in DE2 board, the CPU signals would crash. After we added the synchronizers to the design, the problems had been solved.
2. In this lab, we design and create the CPU components for a simple computer (slc3) and fill the reset of state machines in ISDU. After we finish the above two steps, it is necessary to add synchronizers in the slc3 design, otherwise the CPU will crash. From this task, we master the skills for the integration of modules and complex design using state machines. The long and the short of it, we are glad to learn knowledge in this lab and look forward to meeting next lab task!