Digital System Design: Designing OISC Processor

20170285 박지현

20170337 손소정

20180653 주민규

1. Introduction

From Lab1 to the final project, our team members worked on reducing delay for the first goal. Our plan was reducing the delay from the beginning of the project. Minimizing CPI and clock period, reducing delay for each module, including Labs, and other minor changes were done. Improvements in terms of area reducing was also done by tasks such as assigning modules for repetitive process, reducing the use of modules and changing gates. Lastly, for a processor to have no hazards and glitches, we added more gates, considering the trade-off between area/delay and accuracy.

2. State Assigning

Starting from reset state, there are total 7 states.

- State Diagram (Mealy Machine)

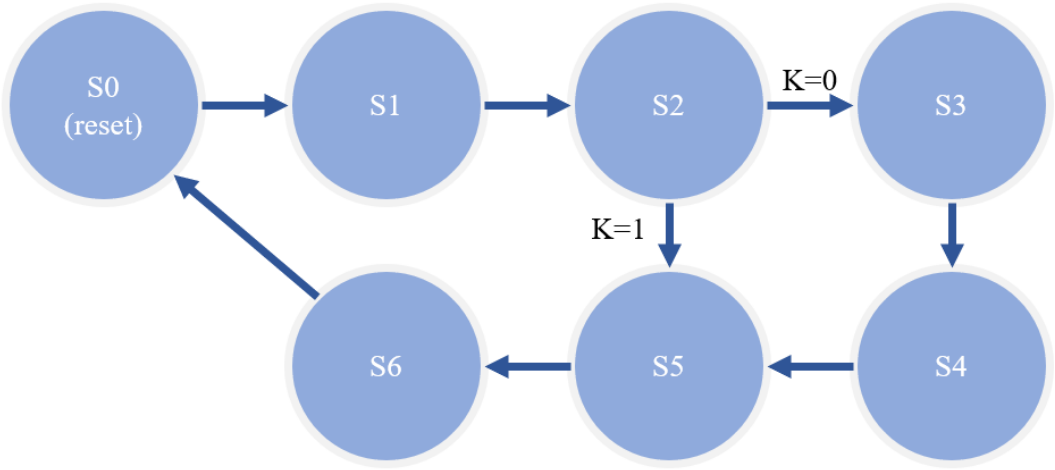

Figure 1 State Diagram

Here, K is a variable which detects if a is equal to b. If a is equal to b, then loading data only once is enough, and neither subtracting **b** from **a** nor calculating **leq** is required, since both of their values are already decided at the point the two addresses are identical.

- State Introduction
  - S0: reset state
  - S1: reading PC – data out is **a**
  - S2: reading PC+1 – data out is **b** / check **a**=**b** and if **a**==**b**, go to S5
  - S3: reading **a** – data out is **mem[a]**, invert every bit of **mem[a]**
  - S4: reading **b** – data out is **mem[b]**, **b'** = **b**-**a** for lower 4 bits
  - S5: **b'** = **b**-**a** for higher 4 bits
  - S6: reading PC+2 – data out is **c**, if **leq**=1(**b**<=0), branch **c** to PC
  - S0-1: next instruction, write **b'** at **b**

state	register a	register b	sram_dout
s0			
s1			a_addr
s2	a_addr		b_addr
s3	a_addr	b_addr	a_data
s4	a_data	b_addr	b_data
s5	b_data	b_addr	
s6	b-a	b_addr	c

Figure 2 Data stored in Registers and Bus

- State Table

state		next state K=0	next state K=1	output sram_en	sram_we	addr_sel	en_a	en_b_addr	pc_inc	pc_br
s0	000	s1	s1	1	0	0	0	0	1	0
s1	100	s2	s2	1	0	0	1	1	0	1	0
s2	001	s3	s5	1	0	1	0	0	1	0	0
s3	111	s4	-	1	0	0	1	1	0	0	0
s4	110	s5	-	0	0	0	1	1	0	0	0
s5	101	s6	s6	1	0	0	1	1	0	1	0
s6	011	s0	s0	1	1	1	0	0	0	0	leq

Figure 3 State Diagram

- Evolution

At the first step of designing the states, there were some misunderstanding of the states' output and the real function of the states. Hence, we had 10 states and 3 inputs, which led us to solve 7-input Karnaugh map through the help of the internet. However, just the day before the interim presentation, our group found out that we misunderstood the outputs and the functions and therefore we could revise it all from the beginning.

The next design had 7 states, same as the given one in the pdf (load a, load b, load mem[a], load mem[b], and finally load c). This time, we figured out the way to delete the register c and reduce to 6 states. Therefore, our interim presentation was not done (since we did all this at the dawn of the presentation day) and we could not present the result but estimated the average CPI as 5.

The final design has more states, but we took an idea from a group from their presentation to divide the subtracting process for two clock periods. Therefore, if the subtracting process is not needed, we can jump two states.

3. File and Module Information

- gates.v
- sram_sample.v

  Our group received the sample sram file.
- sram_array.v
- controlpath_module.v

  = fsm (only)
- datapath_module.v

  Includes modules used in processor_top

  - register

    : We made the register through flip flop. We used this twice in one instruction.
  - program_counter

    : Rather than clock gating, we controlled the flip-flop input d.
  - mux_3

    : 3 1-bit input to 1 1-bit output with 2-bit selection input
  - mux_2

    : 2 1-bit input to 1-bit output with 1-bit selection input
  - mux_bus_24_to_8

    : This mux is used to choose the input address for the pc counter. 00 equals to **PC**, 01 equals to **register_a** and 10 equals to **register_b_addr**. The selection input is **addr_sel**, which is 2 bits. We used **mux_3** 8 times.

- mux_8 & mux_1

  : Their role is to maintain the data until the new data comes in (like the register). The difference between register is since latches have big delay and area, we store data by mux. The input of the mux is 1) a new data we want to store 2) the output of the mux. This way we can receive the new data when we want. We receive the new data only for the state we want, and when the clock is zero. This way, we avoid glitches, since the timing of clock signal and the states' names are different.

- adder_4

  : For the subtracting process, we inverted the input of mem[a] before, so we just used the adder. Also, as we planned to divide the adding process into 2 clock periods, we only needed 4-bit adder. We did not forget to store the c4 (carry of the 4^{th} bit), and the c0 equals to 1.

- less_than_or_equal_to_zero

  : This module is needed to get the result of **leq** through **b'**.

- compare

  : Its role is to compare two 8-bit data. If the two inputs are identical, the result is 1, and if not, the result is 0.

- processor_top.

  : We arranged the codes in the order of instruction. At first, we used the module sram and then declared two registers for **register_a** and **register_b_addr**. As we enter state1, we store the **sram_dout**, which is **a_addr**, into **temp_a**. At state 2, we store **b_addr** into **temp_b**. This way we can compare **a** and **b**. Next, we have the pc counter and mux bus to choose

which address will go into pc counter. Then, at state3, we store the inverted version of **a_data** into **a_inv**, and at state4, we store **b_data**. After calculating **b'** we calculate **leq** and store this data into **temp_leq** by **mux_1**. Lastly, we have the fsm.

4. Simulation Results

- fibonacci.asm

```
#  34( 34)
# --------------------------
#      31305: Halt by beep
```

- gcd.asm

```
#  99( 99) 27( 27)
#  27( 27) 99( 99)
#  18( 18) 27( 27)
#   9(  9) 18( 18)
#   9(  9)
# --------------------------
#      10807: Halt by beep
```

- gcd_fast.asm

```
#  99( 99) 27( 27)
#  27( 27) 99( 99)
#  18( 18) 27( 27)
#   9(  9) 18( 18)
#   9(  9)
# --------------------------
#       9690: Halt by beep
```

- hello_world.asm

```
# Hello, world!!
#
# --------------------------
#      32829: Halt by beep
```

5. Minimum Clock Period and Total Area

- Minimum Clock Period

  25.4

- Total Area

	#	area	total
sram	1	37.25	37.25
sram_array_64_32	1	0	0
register	2	56	112
mux_8	5	44	220
compare	1	22.5	22.5
program_counter	1	148.25	148.25
mux_bus_24_to_8	1	88	88
adder_4	2	51.25	102.5
less_than_or_equal_to_zero	1	9.25	9.25
mux_1	1	5.5	5.5
processor_fsm	1	41	41
gates of top			31.75
TOTAL			818

Figure 4 Area Calculation

6. Group Member Participation

Our group members took similar amounts of tasks.

지현 did the speedy coding and organize the files and modules. She also came up with an idea that we should first try with higher level implementation (using assign) and then changing to lower level, which is appropriate to our assignment. She checked if our program runs well with the correct result, and lastly checked the time and tried to reduce it. She made the interim ppt slides, too.

소정 did the overall design of the states, organized it and revised it several times whenever she thought she had to. She helped 지현 do the coding in the way which she thought. She came up with the idea to make more inputs and outputs of the fsm in order to make the program have less states and glitches also. She worked for the reports, too.

민규 was the one who was the most meticulous, so his codes for lab1 through 3 was used in the program. Even though we three always exchanged the idea for the lab, he had the least delay and area, despite the same idea as 소정 and 지현. He is good at criticizing 소정's plans so she could develop her design plans more concrete as she got advice from 민규. 민규 also worked to reduce the delay and area, especially for the subtractor part, since it was is made by him.

The participation rate was decided as 3 for 지현, 4 for 소정, and 3 for 민규.

7. Finishing the Project

Our group members agreed that this was an extraordinary class. It was hard to figure out what the lecture really means, and we had a tough time starting the final project. Finally, we could make a processor of our own and we figured out that everyone has the potential, and it's the matter of time. Our group spent a lot of time together to come up with a better idea despite other assignments from other lectures were also risky. As the end of the semester is coming, we felt that we had improved a bit through this project.