EECS2070 02 Exam 1

15:30 ~ 17:30, November 2, 2021

➢ Instructions

1. There are **three (3) design problems** in this exam, with the PDF specification of **nine (9) pages** in total.
   - You also have the hardcopy of the first page to sign and hand back.
2. Download the **`*.cpp`** files from OJ. Change them to **`*.zip`** and decompress them. (Skip the `*.h` files. The OJ enforces having a `*.h` file.)
3. Submit each of your Verilog code to OJ **immediately** after it is done.
   a. You have the responsibility to check if the submission is successful.
   b. The module names should be **exam1_A**, **exam1_B**, and **exam1_C**.
   c. The first line of each Verilog code should be a comment with your student ID and name as follows:

```
// 107123456 王小明
module exam1_A (...
```

   d. The **exam1_C.v** should be able to be compiled by Vivado, generating the bit file.
   e. **The submission is due at 17:30**!
4. The score will get deducted if you fail to follow the rules.
5. Hand back this problem sheet **with all the following items checked**. Also **sign your name and student ID**.

☐ I confirm that I follow the naming rule of the modules. And the first line of each answer code shows my student ID and name.

☐ I confirm that all my answers are submitted successfully.

- ✓ Submit the module **exam1_A** and the complete design for Problem A;
- ✓ Submit the module **exam1_B** and the complete design for Problem B;
- ✓ Submit the module **exam1_C** and the complete design for Problem C.

☐ I hereby state that all my answers are done on my own.

ID: _____ Name: _____

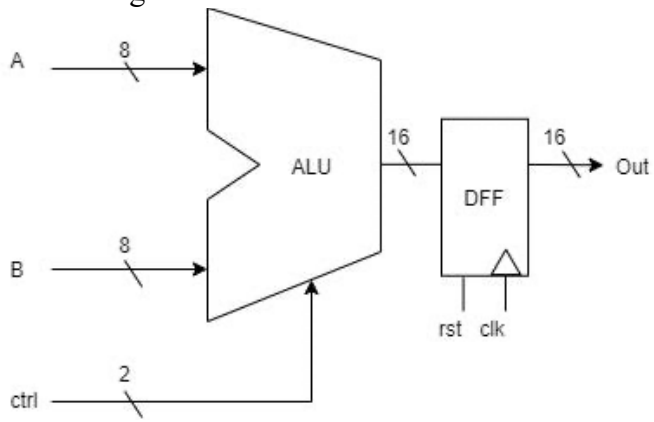
➢ Design Problems

A. [30%] [Verilog simulation]

1. To prevent timing issues when doing implementation (APR), it is recommended to add a flip flop before the output port. Thus, you are going to design a **synchronous** ALU in this problem.
2. Your design should be reset **synchronously**.
3. Your design should change its value at the **positive edge** of each clock cycle.
4. You must complete the Verilog template **exam1_A.v** with the given testbench and pattern file, **exam1_A_tb.v**, **pattern_A.dat**.
   (Refer to the **appendix** at the end to know how to add the pattern files to the simulation sources.)
   Pass Message:

```
Function Multiplication PASS! 6/6
Function Append         PASS! 6/6
Function Shift          PASS! 6/6
Function Default        PASS! 6/6
Pattern Score:          24/24
```

5. Block diagram



6. This ALU has signals below.

Signal Name	I/O	Bit-width	Description
clk	Input	1	Clock (positive-edge triggered).
rst	Input	1	**Synchronous** active high reset
A	Input	8	Signed ALU source
B	Input	8	Signed ALU source
ctrl	Input	2	ALU control signal
out	Output	16	Signed ALU output

7. ALU function table:

Name	ctrl [1:0]	Function
Function 1	2'b00	out = A * B with sign extension.
Function 2	2'b10	out = (A & B) append (A|B).
Function 3	2'b11	out = Arithmetic right shift A by 3 bits with sign extension.
Function 4	Others	out = A - B with sign extension.

8. Example:
   (1) (CYCLE1) ctrl = 2'b00, A = 8'd12, B = 8'd44
   (CYCLE2) out = 16'd528
   (2) (CYCLE1) ctrl = 2'b10, A = 8'b0000_0001, B = 8'b0000_0011
   - A&B = 8'b0000_0001
   - A | B = 8'b0000_0011

   (CYCLE2) out = 16'b0000_0001_0000_0011
   (3) (CYCLE1) ctrl = 2'b11, A = 8'b0011_1100, B = 8'b0000_1111

(CYCLE2) out = 16'b0000_0000_0000_0111

(4) (CYCLE1) ctrl = 2'b01, A = 8'd55, B = 8'd44
(CYCLE2) out = 16'd11

9. Grading

Name	Score
Function 1	6%
Function 2	6%
Function 3	6%
Function 4	6%
Synchronous reset	3%
Clock triggered event at positive edge	3%

10. Note
    (1) You cannot modify testbench or patterns, TA will use the same testbench and different patterns to test your design.
    (2) The grading is based on correctness. Pass/fail messages of the testbench are only to assist the debugging.
    (3) Take care of the **signed numbers**.

B. [30%] [Verilog simulation]

1. Implement a specific counter with the following rules.
2. You must complete the Verilog template **exam1_B.v** with the given testbench and pattern files, **exam1_B_tb.v**, **pattern_B.dat**.
   (Refer to the **appendix** at the end to know how to add the pattern file to the simulation sources.)
3. Your design should be reset **synchronously**.
4. Your design should change its value at the **positive edge** of each clock cycle.
5. Make sure you pass the simulation with the following PASS message:

```
Count Up    10/10!
Count Down  10/10!
Count Again 10/10!
Your total score:          30/30
```

6. IO signals and description:

Signal	I/O	Description
clk	Input	Clock signal (positive-edge triggered).
rst	Input	Synchronous active-high reset.
result[19:0]	Output	The (current) value of the counter

7. The counter will count upward from **0** to **183921** and then count downward to **0 repeatedly**.
8. The counter will follow the rules:
   (1) When counting upward:
   Let the current value be **S.** Its initial value is **0**.
   $\mathbf{S_i}$ will increase by the following rules when it is the **i**$^{\mathbf{th}}$ count up.
   Note that **i starts from 1, not 0**.
   (a) When **S**%8 is equal to **i**%8, **S** will increase by **i*3**.
   (b) When **S**%8 is not equal to **i**%8, **S** will increase by **i.**
   (2) When counting downward:
   Let the current value be **S**.
   S will decrease by **i**$^{\mathbf{th}}$ when it is the **i**$^{\mathbf{th}}$ countdown:
   S ➔ S – 1 ➔ (S – 1) – 2 ➔ (S – 1 – 2) – 3 ➔ … ➔ 0
9. Your output should look like the following sequences:
   (1) Counting upward
   0 ➔ 1 ➔ 3 ➔ 12 ➔ ... ➔ 182870 ➔ 183395 ➔ 183921
   S_1 S_2 S_3 S_4 S_{525} S_{526} S_{527}
   (2) Counting downward
   183921 ➔ 183920 ➔ 183918 ➔ ... ➔ 1211 ➔ 606 ➔ 0
   (3) Consecutive counting upward and downward
   0 ➔ 1 ➔ ... ➔ 183395 ➔ **183921** ➔ 183920➔ ... ➔ 606 ➔ **0** ➔ 1 ➔ 3 ➔ …
   Note that 183921 (or 0) appears only once when reaching the maximum (or minimal) value.
10. **Design Constraint: You must NOT use a lookup table or a fixed, hand-coded sequence to generate the counter values.** Otherwise, your design will get a zero score.
11. Grading

Name	Score
Count-up	10%
Count-down	10%
Count-again	10%

12. Note
    You cannot modify testbench or patterns, TA will use the same testbench and different patterns to test your design. The grading is based on correctness. Pass/fail messages of the testbench are only to assist the debugging.

C. [40%] [FPGA Implementation]

1. Use the Verilog template **exam1_C.v** to implement the LED controller. DO NOT modify IO signals of **exam1_C**.
2. The LED light moving is synchronous with the clock rate of 100MHz / 2^{25}.
3. The state machine is synchronous with the clock rate of 100MHz / 2^{13}
4. The seven-segment display is synchronous with the clock rate of 100MHz / 2^{15}
5. Your design should be reset **synchronously**.
6. Your design should change its value at the **positive edge** of each clock cycle.
7. Here is the table showing the function with the I/O connection:

Name	I/O	Pin	Description
clk	Input	W5	100MHz clock signal
rst_btn	Input	BTNC	Active-high reset
start_btn	Input	BTNL	Start/Pause the movement
sw	Input	16-Switches	Enable walls for collision
DIGIT [3:0]	Output	4-Digits Pin	Display state and score
DISPLAY[6:0]	Output	7-Segment Pin	Display state and score
led [15:0]	Output	16-LEDs	Display Mr.1, Mr.2, and walls

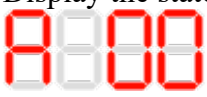
8. Please refer to the demo video (**exam1_C_DEMO.mp4**) for further details.
9. Function description
   (1) There are two LED runners, called Mr.1 and Mr.2.
   (2) Mr.1 and Mr.2 race on the 16 LEDs of the FPGA Demo board.
   (3) There are three states: **RESET**, **START**, and **PAUSE**.
   - RESET
     Mr.1 and Mr.2 are waiting for the start signal (press **start_btn**) to start racing on LEDs at the initial positions.
     (a) State transition
     For every state, if **rst_btn** has been pressed, the FPGA should enter RESET state to initialize itself immediately. And if the current state is RESET, you can press **start_btn** to enter **START** state to start racing.
     (b) Switches
     Switches can enable walls, and also light up the corresponding LEDs.
     (c) LED
     LEDs will be turned on to represent Mr.1, Mr.2, and walls.
     (d) Example
     - Mr.1's initial position is at LED 1.
     - Mr.2's initial position is at LED 10.
     - No switch is enabled.

     The LEDs: (●: Mr.1, ●: Mr.2, ●: Wall, ○: LED off)
     (LED15) ○○○○○●○○○○○○○●○ (LED0)

     - If switch 4 is enabled in RESET state, LED 4 will light up.

     The LEDs will be: (●: Mr.1, ●: Mr.2, ●: Wall, ○: LED off)
     (LED15) ○○○○○●○○○○○●○○○●○ (LED0)

     (e) Seven-segments
     Display the state "R" at the leftmost digit. The two rightmost digits should be zero.

   - START
     Mr.1 and Mr.2 start racing on the LEDs. They will collide with each other or a wall. If the collision happens, the racer will change to the opposite direction.

There is a counter display on the two rightmost digits of the seven-segment, recording the collision times of Mr.1 and Mr.2 (**not including wall collision**).

(a) State transition

If **start_btn** been pressed or the recorded collision number reaches **15**, the state will be changed to PAUSE.

(b) Switch

A switch can enable a wall. Mr.1 and Mr.2 will collide with a wall. The racers **can be stuck between two walls.**

(c) LED

LEDs will light up to represent Mr.1, Mr.2, and walls.

(d) Seven-segment display

Display the state "S" at the leftmost digit, and the number of collisions at the two rightmost digits.

E.g., the number of collisions is 13

(e) Situation 1

- Mr.1's initial position is at LED 1. Its direction is LEFT.
- Mr.2's position is at LED 10. Its direction is RIGHT.

Cycle0: (●: Mr.1, ●: Mr.2, ●: Wall, ○: LED off)

(LED15) ○○○○○●○○○○○○○○●○ (LED0)

-> <-

Cycle1:

(LED15) ○○○○○○●○○○○○○●○○ (LED0)

-> <-

Cycle2:

(LED15) ○○○○○○○●○○○○●○○○ (LED0)\

-> <-

Cycle3:

(LED15) ○○○○○○○○●○○●○○○○ (LED0)

-> <-

Cycle4: (Racer collision, recorded collision times increase by 1)

(LED15) ○○○○○○○○○●●○○○○○ (LED0)

-><-

Cycle5:

(LED15) ○○○○○○○○●○○●○○○○ (LED0)

<- ->

(f) Situation 2

When a racer reaches the rightmost LED (LD0), it will appear at the leftmost LED (LD15) afterward, and vice versa.

Cycle X: (●: Mr.1, ●: Mr.2, ●: Wall, ○: LED off)

(LED15) ○○○○○○○●○○○○○○○● (LED0)

<- ->

Cycle X+1:

(LED15) ●○○○○○●○○○○○○○○○(LED0)

-> <-

(g) Situation 3
When a wall at LED 4 is set, the racer will collide with it.
Cycle X: (●: Mr.1, ●: Mr.2, ●: Wall, ○: LED off)
(LED15) ○○○○○○○●○○●●○○○○ (LED0)
<- ->
Cycle X+1:
(LED15) ○○○○○○●○○●○●○○○○(LED0)
<- <-

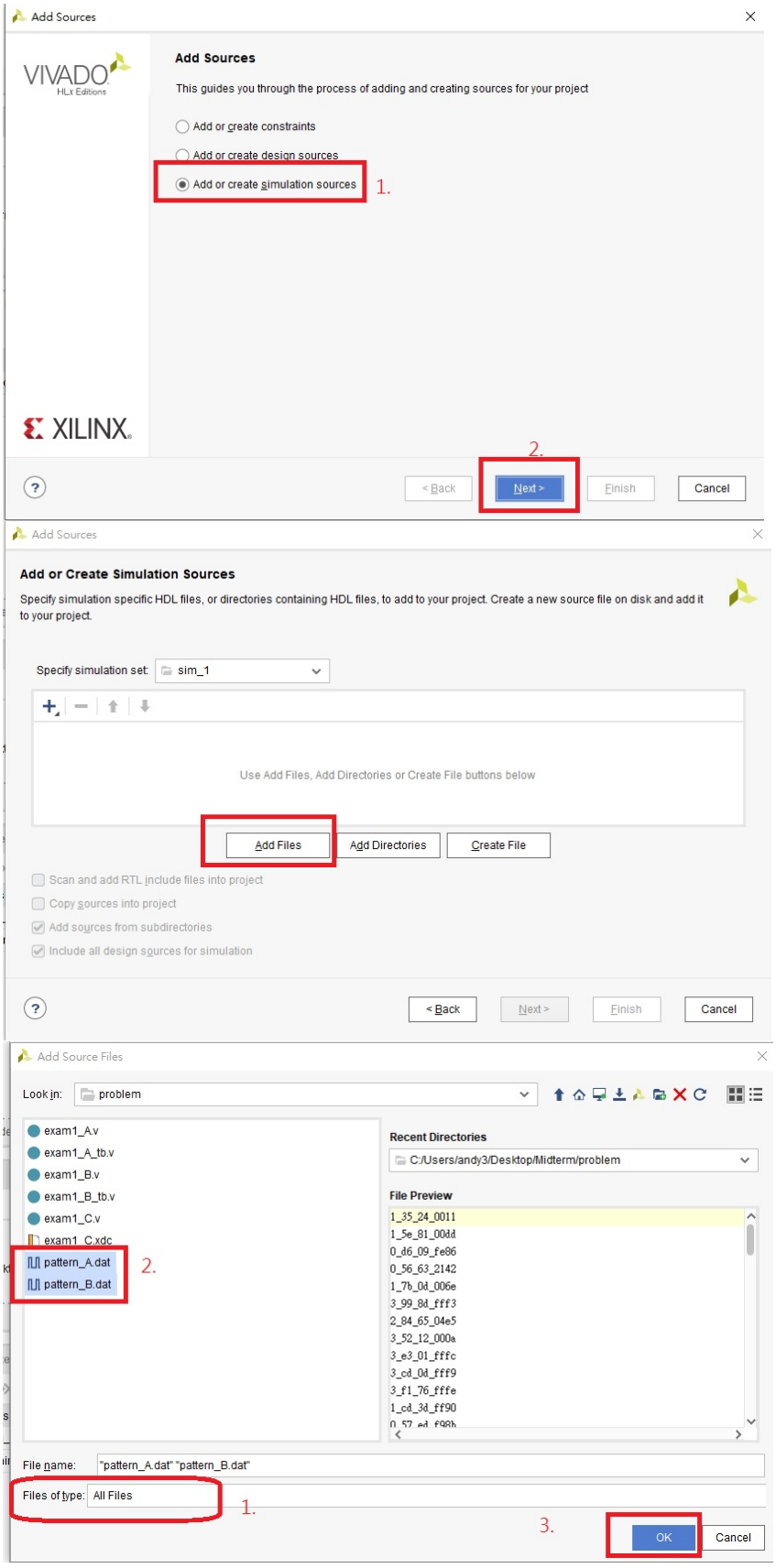
(h) Situation 4
Racer stuck by walls at LED 4 and LED 6.
Cycle X: (●: Mr.1, ●: Mr.2, ●: Wall, ○: LED off)
(LED15) ○○○○○○○●○●●●○○○○ (LED0)
<- ->
Cycle X+1:
(LED15) ○○○○○○●○○●●●○○○○(LED0)
<- <-

- PAUSE
Mr.1 and Mr.2 stop moving.
(a) State transition
If **start_btn** been pressed again, the state will change to **START**, and start racing.
But **if the number of collisions is already 15, the state should stay at PAUSE**.
(b) Switch
A switch can enable a wall. Mr.1 and Mr.2 will collide with a wall. The racers **can be stuck between two walls.**
(c) LED
LEDs will light up to represent Mr.1, Mr.2, and walls.
(d) Seven-segment display
Display the state "P" at the leftmost digit, and the number of collisions at the rightmost two digits.
E.g., the number of collisions is 7

10. Refer to the XDC constraint file (**exam1_C.xdc**) to know the pin connection first. Also, DO NOT modify the constraint file. Otherwise, your design will fail the synthesis then get a ZERO score.
11. There are already several modules in the template, including clock divider, seven-segments, debounce, and one-pulse modules. You can modify them if necessary. You must include **all the modules** you need in the same file and submit it, or you will get a 5 points penalty.
12. Grading

Function	Score
Reset	5%
LED: Racer movement	10%
LED: Racer collision	10%
LED: Wall enable & collision	5%
Seven-segment: State	5%
Seven-segment: Collision times	5%

Appendix: How to add pattern.dat to the simulation sources.



Happy Designing!

(If you have too much time left, there is always a joke for you.)

One day, a mechanical engineer, an electrical engineer, and a computer engineer drove down the street in the same car when it broke down.
The mechanical engineer said, "I think the engine broke. We have to fix it."
The electrical engineer said, "I think there was a spark and something's wrong with the electrical system. We have to fix it."
Both of them turned to the computer engineer and asked, "What do you think?"
The computer engineer replied, "I have no idea what happened. But why don't we close all the windows, and then open the windows again? That always works for me!!"