EECS2070 02 Exam 1

15:30 ~ 17:30, October 25, 2022

I. Instructions

1. There are **three (3) design problems** in this exam, with the PDF specification of **eight (8) 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 that there must be a `*.h` file.)
3. Submit each Verilog file to OJ **immediately** once 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 file should be a comment with your student ID and name as follows:

```
// 110123456 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. Please **take the OJ password slip with you** when leaving your seat. Do not litter!
5. The score will get deducted if you fail to follow these rules.
6. Hand back this problem sheet **with all the following items checked**. Also, **sign your name and student ID.**

☐ I confirm that I read the instructions carefully and understand that my score will get deducted if failing to follow these rules.

☐ 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, if finished, 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 understand that the generated bit file will be scored. And the incorrect submission will result in a zero score.

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

ID: _____ Name: _____

II. Design Problems

A. [30%] [Verilog Simulation]

1. In this problem, you are going to design a **synchronous** ALU. To prevent timing issues, you have to add a flip-flop (FF) before the output port (the output should be delayed for 1 cycle).
2. Your design should change its value at the **positive edge** of each clock cycle and be reset **synchronously**.
3. When the reset is triggered, the output should become 0.
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 learn how to add the pattern files to the simulation sources.)
5. Make sure you pass the simulation with the following PASS message:

```
Function 1              PASS!
Function 2              PASS!
Function 3              PASS!
Function 4              PASS!
Pattern Score:          28/28
```

6. IO signals and description:

Signal Name	I/O	Description
clk	Input	Clock (positive-edge triggered)
rst	Input	**Synchronous** active-high reset
A [7:0]	Input	Signed ALU input
B [7:0]	Input	Signed ALU input
ctrl [1:0]	Input	ALU control signal
out [15:0]	Output	Signed ALU output

7. control signal:

Name	ctrl	Function
Function 1	2'b00	out = A*(-B) with sign extension
Function 2	2'b01	out = concatenation of (A bitwise-xor B) and (A bitwise-nor B)
Function 3	2'b11	out = A + B with sign extension
Function 4	Otherwise	If (A<B), out = -1 with sign extension else, out = 1 with sign extension

Examples:

(1) (CYCLE1) **ctrl** = 2'b00, **A** = 8'd5, **B** = -8'd6
(CYCLE2) **out** = 16'd30

(2) (CYCLE1) **ctrl** = 2'b01, **A** = 8'b0110_0110, **B** = 8'b0100_0111
(CYCLE2) **out** = 16'b0010_0001_1001_1000

(3) (CYCLE1) **ctrl** = 2'b11, **A** = 8'd8, **B** = 8'd9
(CYCLE2) **out** = 16'd17

(4) (CYCLE1) **ctrl** = 2'b10, **A** = 8'd8, **B** = 8'd6
(CYCLE2) **out** = 16'd1

8. Grading

Name	Score
Function 1	8%
Function 2	6%
Function 3	8%
Function 4	6%
Synchronous reset	1%
Clock triggered event at positive edge	1%

9. Note
   (1) You cannot modify the testbench or patterns. TA will use the same testbench with other 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) Be aware 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 learn how to add the pattern file to the simulation sources.)
3. Your design should change its value at the **positive edge** of each clock cycle and be reset **synchronously**.
4. Make sure you pass the simulation with the following PASS message:

```
Count Up                PASS!
Count Down              PASS!
Count Again             PASS!
Pattern Score:          30/30
```

5. IO signals and description:

Signal	I/O	Function
clk	Input	Clock signal (positive-edge triggered).
rst	Input	**Synchronous** active-high reset
out [15:0]	Output	The (current) value of the counter

6. The counter will count upward from **1** to **65176** and then count downward to 0 repeatedly.
   Let i denodes the i-th counting step. The counter will follow the rules:
   (1) c_i**= current counting value.**
   (2) c_{i-1}**= previous counting value.**
   (3) Let $S = c_{i-1} + c_i$.

- **Counting upward:** ("/" operator is prohibited in your Verilog code)
  Initially, i **starts with 1. Let** c_0 **= 0 and** c_1 **= 1.**

$$c_{i+1} = \begin{cases} S/_2 \times 3, \text{when } S\%4 = 0 \text{ and } S < 16384, \\ c_i + i, \text{when } S\%4 = 0 \text{ and } S \geq 16384, \\ c_i + 4, \text{when } S\%4 \neq 0 \text{ and } i \text{ is odd}, \\ c_i + 1, \text{when } S\%4 \neq 0 \text{ and } i \text{ is even}. \end{cases}$$

- **Counting downward:** ("%" operator is prohibited in your Verilog code)
  Initially, i **starts with 1. Let** c_0 **= 64752 and** c_1 **= 65176.**

$$c_{i+1} = \begin{cases} c_i \gg (i\%4), \text{when } S\%8 = i\%8, \\ c_i - (i \times 6), \text{when } S\%8 \neq i\%8. \end{cases}$$

※If c_{i+1} will be negative, set $c_{i+1} = 0$ and end the downward counting sequence.

7. Your output should look like the following sequences

	i	1	2	3	4	5	…				
(Upward)	c_i	1	→ 5	→ 6	→ 10	→ 24	→ …	→ 64747	→ 64748	→ 64752	→ 65176
(Downward)	c_i	65176	→ 65170	→ 16292	→ 16274	→ 16250	→ …	→ 539	→ 311	→ 77	→ 0

The counter counts repeatedly: **1→5→…→ 64752 → 65176 → 65170 → … → 77 → 0 → 1 → 5 …**

8. **Design Constraint:**
   (1) You must NOT use a lookup table or a fixed, hand-coded sequence to generate the counter values. Otherwise, you will get a zero score.
   (2) Using the division operator "/" is forbidden in count-up function and will only get a half score.
   (3) Using modulo operator "%" is forbidden in count-down function and will only get a half score.

9. Grading

Function	Score
Count-up	10%(without using "/" operator) 5% (with using "/" operator)
Count-down	10% (without using "%" operator) 5% (with using "%" operator)
Count-again	10%

10. Note

(1) You cannot modify the testbench or patterns. TA will use the same testbench with other patterns to test your design.

(2) The grading is based on correctness. Pass/fail messages of the testbench are only to assist the debugging.

C. [40%] [FPGA Implementation]

1. Complete the Verilog template, **exam1_C.v**, to implement the LED controller. DO NOT modify the IO signals.

2. There are two players, player_AI and player_user. They are in a contest to grab the flag first, one flag for one point. The player who gets 10-point first will win the contest.

3. Your design should change its value at the **positive edge** of each clock cycle and be reset **synchronously**.

4. Here is the table showing the function with the I/O connection:

Name	**I/O**	**Pin**	**Description**
clk	Input	W5	100MHz clock signal
rst	Input	U18 // btnC	**Synchronous** active-high reset
en	Input	T17 // btnR	To enable the movement
set	Input	W19 // btnL	To add the flag where the **sw** indicates
up	Input	T18 // btnU	player_user will become a bigger size with a slower speed
down	Input	U17 // btnD	player_user will become a smaller size with a faster speed
sw [15:0]	Input	16-switchs	To decide where to add the flag
DIGIT [3:0]	Output	4-digits	To control the 7-segment digits
DISPLAY [6:0]	Output	7-segment	To control the 7 segments of a digit
led [15:0]	Output	LEDs LD15-LD0	To show the position of player_AI, player_user, and flags

5. Please refer to the demo video (**exam1_C.mp4**) for further details.

6. Pushing the rst button at any time will enter the **RESET** state.

7. **RESET:**

(1) When being reset, player_AI is on LD15, player_user is on LD2~LD0 with mode1 (explained at rule 8.3).

(LD15) ●OOOOOOOOOOOO●●● (LD0)

(2) The mode of player_user cannot be changed in this state.

(3) rules of adding flags

a. Turn on the switches where you want to add flags.

b. The flag will be added after pushing the **set** button.

c. At most one flag can be set at each position.

d. These flag rules also apply in the **START** and **PAUSE** states.

(4) Pushing the **en** button will enter the **START** state.

8. **START:**
   (1) player_AI moves at a constant speed, one LED at a time with the clock frequency of 100MHz/2^{25}.
   (2) player_AI rotates from left to right initially (i.e., its position after LD0 is LD15). If player_AI collides with player_user, player_AI will change its direction.
   (3) player_user has three moving modes.
      a. mode0: player_user's size is one LED, moving with the clock frequency of 100MHz / 2^{24}.
      b. mode1: player_user's size becomes three LEDs, moving with the clock frequency of 100MHz / 2^{25}.
      c. mode2: player_user's size becomes five LEDs, moving with the clock frequency of 100MHz / 2^{26}.
      d. Pushing the **up** button will enter a higher mode (e.g., mode0 to mode1). Nothing will happen if player_user is in mode2.
      e. Pushing the **down** button will enter a lower mode (e.g., mode2 to mode1). Nothing will happen if player_user is in mode0.
      f. When changing to a higher mode, player_user's size expands at both sides; when changing to a lower mode, player_user's size shrinks at both sides.

      mode0: ○ ○ ○ ○ ○ ○ ● ○ ○ ○ ○
      ⇨ mode1: ○ ○ ○ ○ ○ ● ● ● ○ ○ ○
      ⇨ mode2: ○ ○ ○ ○ ● ● ● ● ● ○ ○
      ⇨ mode1: ○ ○ ○ ○ ○ ● ● ● ○ ○ ○
   (4) player_user rotates from right to left (i.e., its position after LD15 is LD0).
   (5) Once any player touches the flag, that player gets one point and the flag will be removed. You can add the flag again with rule 7.3.
   (6) If two players touch the flag at the same time, player_AI will get the point.
   (7) Don't need to consider the case that player_user touches two or more flags when it is expanding, TA won't test this in this exam.
   (8) The two leftmost digits of the 7-segment display indicate the score of player_AI. The two rightmost digits of the 7-segment display indicate the score of player_user.
   (9) Pushing the **en** button will enter the **PAUSE** state.
   (10) If any player gets 10 points, go in the **FINISH** state.
9. **PAUSE:**
   (1) player_AI and player_user stay in their position.
   (2) Pushing the **en** button will go back to the **START** state.
10. **FINISH:**
   (1) If player_AI wins, the 7-segment display shows "AIAI" on the 7-segment.
   (2) If player_user wins, the 7-segment display shows "USER" on the 7-segment.

   (3) Flashing the LEDs in the FINISH state with the clock frequency of 100MHz / 2^{24}.

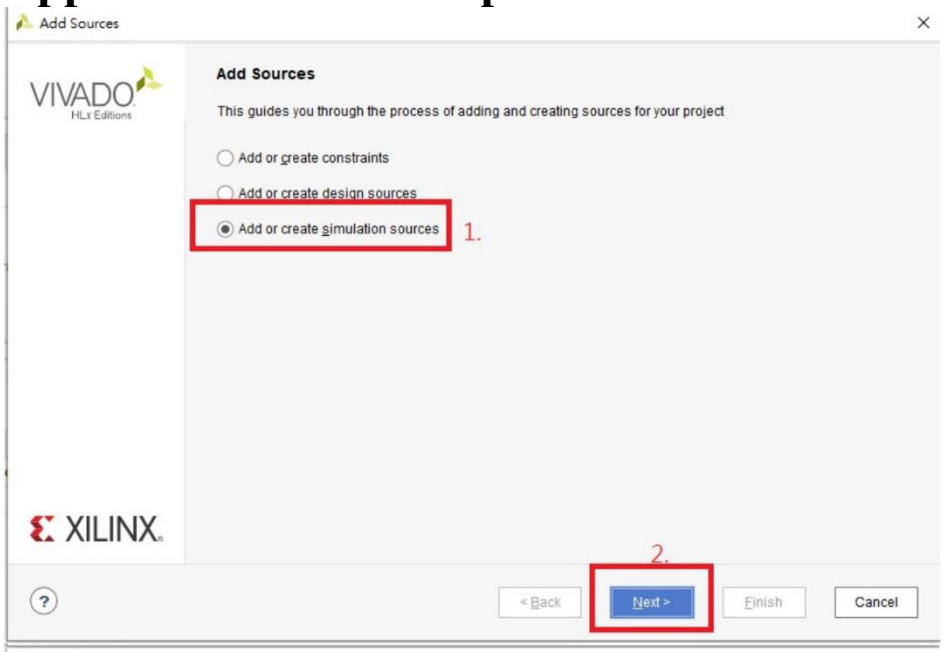
      (LD15) ○●○●○●○●○●○●○●○● (LD0)
      ⇨ (LD15) ●○●○●○●○●○●○●○●○ (LD0)
      ⇨ (LD15) ○●○●○●○●○●○●○●○● (LD0)
      ⇨ …
11. DO NOT modify the XDC constraint file (**exam_1_C.xdc**). Otherwise, your design will fail to test and get a ZERO score.

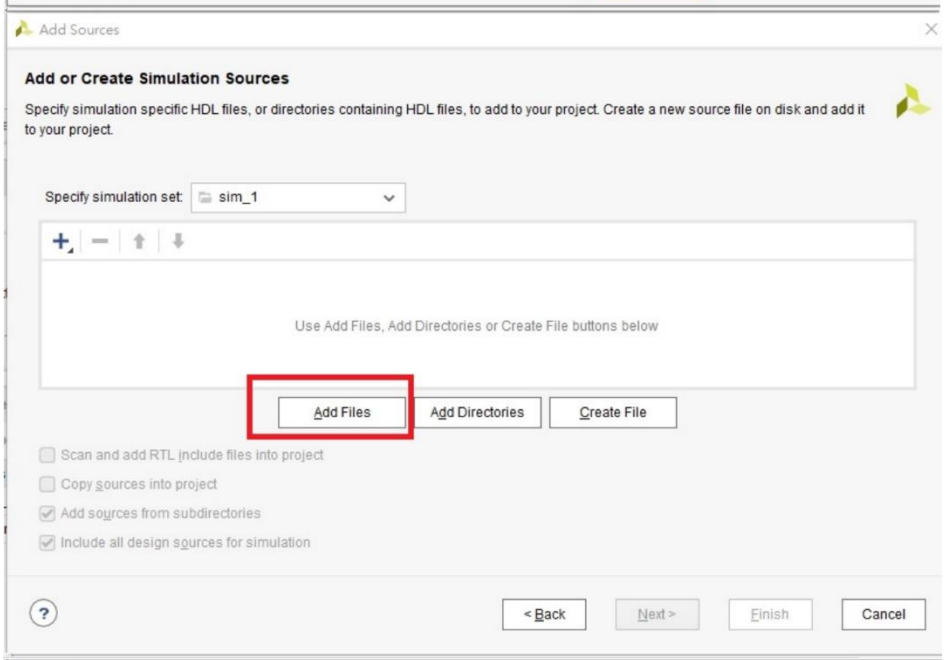
12. There are already several modules in the template, including the clock divider, seven-segments, debounce, and one-pulse modules. You can modify them if necessary. However, the submitted file must be able to generate the bit file, otherwise you will get 0 point in exam1_C.

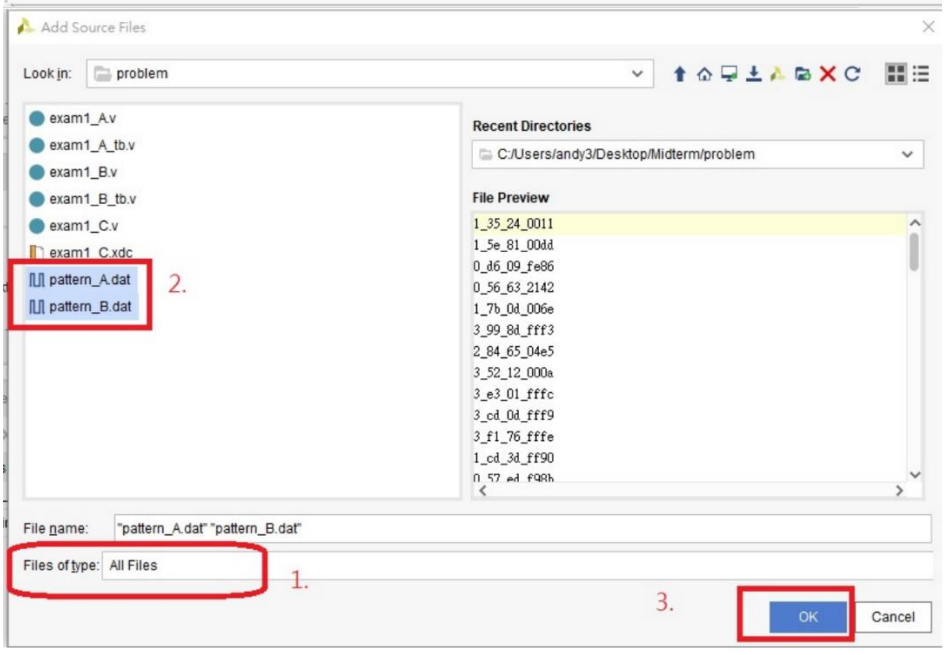
13. Grading

State	Score
RESET	5%
START	20%
PAUSE	5%
FINISH	10%

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!!"