CS210402 Exam 1

15:30~17:30, October 24, 2023

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.

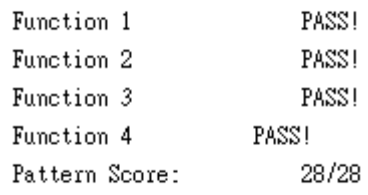
☐ I hereby state that all my answers have been done on my own.

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. The output should become 0 when the design is reset.
4. You must complete the Verilog design in **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+3 with sign extension
Function 2	2’b01	out = concatenation of (A bitwise-and B) and (A bitwise-xor B)
Function 3	2’b11	out = (A + B) << 2 with sign extension
Function 4	Otherwise	If ((A/4) > B), out = 1 with sign extension else, out = -1 with sign extension Note: Don’t use the division operator “/” for this function or you will lose the scores. Be careful with the sign bit and the shifter.

Examples:

(1) (CYCLE1) **ctrl** = 2’b00, **A** = 8’d7, **B** = 8’d5
(CYCLE2) **out** = 16’d38

(2) (CYCLE1) **ctrl** = 2’b00, **A** = 8’d-7, **B** = 8’d5
(CYCLE2) **out** = 16’d-32

(3) (CYCLE1) **ctrl** = 2’b01, **A** = 8’b1110_0010, **B** = 8’b1100_1110
(CYCLE2) **out** = 16’b1100_0010_0010_1100

(4) (CYCLE1) **ctrl** = 2’b11, **A** = 8’d7, **B** = 8’d12
(CYCLE2) **out** = 16’d76

(5) (CYCLE1) **ctrl** = 2’b10, **A** = 8’d8, **B** = 8’d3
(CYCLE2) **out** = 16’d-1

Hint: You can declare the input signal as “signed” to enable signed operations on the signal.
Examples:

```
wire signed [7:0] data;
```

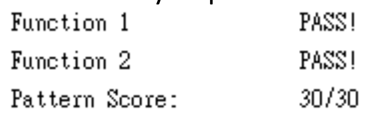
8. Grading

Name	**Score**
Function 1	8%
Function 2	6%
Function 3	8%
Function 4	6% (without using “/”)
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 an error correction design 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
4. Your design should be reset **synchronously**.
5. Make sure you pass the simulation with the following PASS message:



```
Function 1          PASS!
Function 2          PASS!
Pattern Score:      30/30
```

6. IO signals and description:

Signal	**I/O**	**Function**
clk	Input	Clock signal (positive-edge triggered).
rst	Input	**Synchronous** active-high reset
data[11:0]	Input	The encoded input
decoded [7:0]	Output	The decoded data
out [2:0]	Output	The final output

7. Assume that we have a set of 256 input data. Each of them consists of 4 redundant bits (r4, r3, r2, r1) and 8 data bits (d8, d7, d6, d5, d4, d3, d2, d1) (see Table 1). These data are encoded with Hamming code. In this scenario, each data is subject to one single-bit error. In the first step, our objective is to correct these errors and remove the redundant data to get the decoded data.

Bit	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
Notation	d8	d7	d6	d5	r4	d4	d3	d2	r3	d1	r2	r1
data	0	1	1	0	0	0	1	0	1	1	0	0

Table 1

We can define four correction bits as follows to detect the error position:

$c1=B1 \oplus B3 \oplus B5 \oplus B7 \oplus B9 \oplus B11$

$c2=B2 \oplus B3 \oplus B6 \oplus B7 \oplus B10 \oplus B11$

$c3=B4 \oplus B5 \oplus B6 \oplus B7 \oplus B12$

$c4=B8 \oplus B9 \oplus B10 \oplus B11 \oplus B12$

Since we use the **even parity** scheme, the value of each correct bit **should be 0**. If B3 happens to bit-flip during transmission, c1 and c2 will be 1 while c3 and c4 remain 0. Hence, we can identify that B3 has an error, and we can flip its value to correct the error. We list the correction table with all potential combinations for your reference.

{c4, c3, c2, c1}	Error bit
0000	No Error
0001	B1
0010	B2
0011	B3
0100	B4
0101	B5
0110	B6
0111	B7
1000	B8
1001	B9
1010	B10
1011	B11
1100	B12

Next in the second step, we will use the decoded data for the subsequent operations.
The first two MSBs in the decoded data represent the request. (request = decoded[7:6]).
The next three bits in decoded data represent the operand a. (a = decoded[5:3]).
The last three bits in decoded data represent the operand b. (b = decoded[2:0]).
The following table summarizes the operations to execute.

Request	**Operation**
2'b00	out =a and b
2'b01	out =a or b
2'b10	out =a
2'b11	out =b

8. Here are some examples (the red-colored bits are error bits):

Encoded Input	**Decoded Data**	**Final Output (out)**
0100 0010 1100	01100101	101
1000 0000 0000	00000000	000
1010 1100 0000	10111000	111
0101 0010 0111	01010101	111

9. Grading

Function	Score
First step (error decoding)	20%
Second step (final operation)	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 snakes in the cave competing to eat apples, with one point per apple. The competition will last for 10 seconds or will end when the score difference between them is equal to or larger than 3.
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 change the game state
set	Input	W19 (btnL)	To add the apples to the positions that the **sw** indicates
up	Input	T18 (btnU)	To increase the mode level of snake111
down	Input	U17 (btnD)	To decrease the mode level of snake111
sw [15:0]	Input	16-switchs	To indicate the positions to add an apple
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 positions of snake1, snake111, and apples

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, Snake111 is on LD15~13 with mode = 1 (explained at rule 8.5), Snake1 is on LD0.
   (LD15) ●●●○○○○○○○○○○○○● (LD0)
   (2) The snakes will not move in this state.
   (3) The mode of Snake111 cannot be changed in this state.
   (4) Pushing the **en** button will enter the **START** state.
8. **START:**
   (1) The timer will start counting from 0 at the frequency of 100MHz/2^{26} (so the timer counts approximately in seconds).
   (2) Rules of adding apples:
      a. We will only add apples in **START** state.
      b. Turn on the switches where you want to add apples.
      c. The apples will be added after pushing the **set** button. The corresponding LEDs will be on.
      d. At most **one** apple can be set at each position.
      e. These apples can't be placed on the snakes. No effect will occur if you attempt to do so.
   (3) There are some situations for snakes to change their direction.
      a. The snakes reach the right end (LED0) or the left end (LED15).
      b. The snake encounters another snake: the right end of Snake111 and the left end of Snake1 come into contact (the two ends are next to each other or if they overlap each other). Then, Snake111 will turn left and Snake1 will turn right.

      Ex:
      Assume at t=0: when Snake111 at 100MHz/2^{26} and Snake1 at 100MHz/2^{25}. And they are both triggered.
      (LD15) ○○●●●○○●○○○○○○○○ (LD0)
      → ←

t = 1 (100MHz/2^{25}): now they meet each other. Assume that at this point, Snake1's clock is triggered but Snake111's clock isn't yet, but we want both snakes to change their direction for the next step.

(LD15) ooo●●●●ooooooooo (LD0)

← →

t = 2 (100MHz/2^{25}): Both snake's clocks are triggered.

(LD15) ooo●●●o●oooooooo (LD0)

← →

c. The snake will change direction but move according to their clock frequency.

(4) Snake111 moves from left to right initially.

(5) Snake111 has three moving modes.

a. Mode 0: Snake111 moves with the clock frequency of 100MHz / 2^{24}.

b. Mode 1: Snake111 moves with the clock frequency of 100MHz / 2^{25}.

c. Mode 2: Snake111 moves with the clock frequency of 100MHz / 2^{26}.

d. Pushing the **up** button will enter a higher-level mode (e.g., Mode 0 to Mode 1). No action will occur if Snake111 is already in Mode 2.

e. Pushing the **down** button will enter a lower-level mode (e.g., Mode 2 to Mode 1). No action will occur if Snake111 is already in Mode 0.

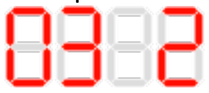
(6) Snake1 moves at a constant speed, one LED at a time with the clock frequency of 100MHz/2^{25}

(7) Snake1 moves from right to left initially.

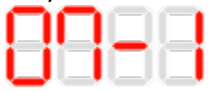
(8) Once any snake touches the apple, that snake gets one point. The apple will be removed (the LED will turn off accordingly). After removal, the same position can be added again.

(9) If two snakes touch the same apple at the same time, Snake111 will get the point.

(10) The two leftmost digits of the 7-segment display indicate the timer.
The two rightmost digits of the 7-segment display indicate the score of Snake111 minus the score of Snake1.
Example:



"03 2" when the timer counts to 3; Snake111 has 6 points and Snake1 4 points (the difference is 2).



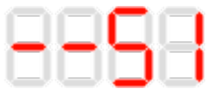
"07-1" when the timer counts to 7; Snake111 has 5 points and Snake1 has 6 points (the difference is -1).

(11) If the timer reaches 10 seconds (i.e., the timer counting from 9 to 10) or the score difference is equal to or larger than 3, go in the **FINISH** state.

9. FINISH:

(1) If Snake111 wins, the 7-segment display shows "S111".

(2) If Snake1 wins, the 7-segment display shows "--S1".
If nobody wins, the 7-segment display shows "----".



Example for "--S1"

(3) Turn on all LEDs.

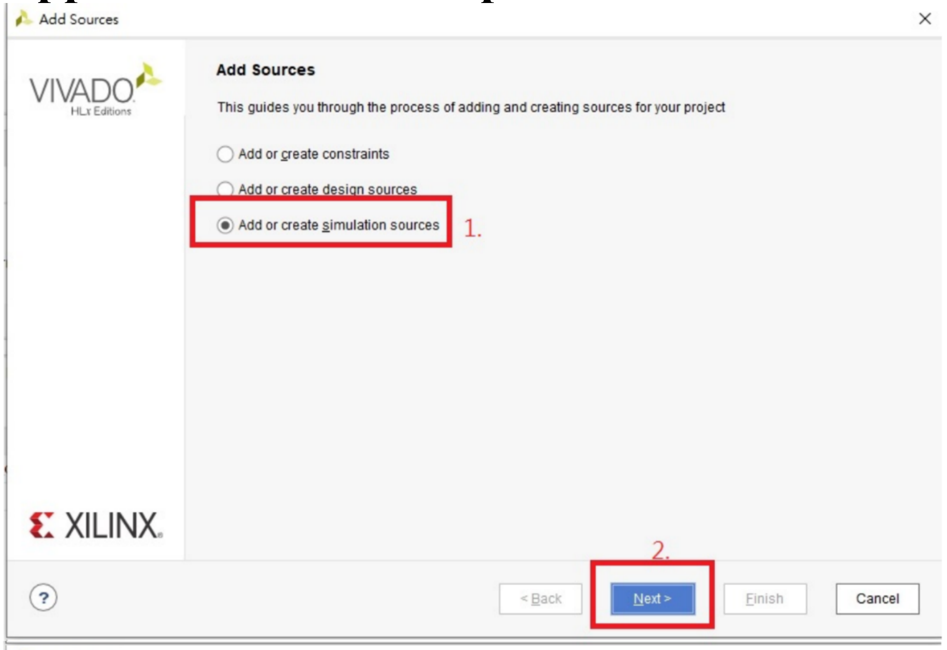
(LD15) ●●●●●●●●●●●●●●●● (LD0)

(4) Stay in the FINISH state until the **en** is pressed. Once the **en** is pressed, go to the **RESET** state.

10. DO NOT modify the XDC constraint file (**exam_1_C.xdc**). Otherwise, your design will fail to test and get a ZERO score.
11. There are already several modules in the template (**exam1_C. v**), 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.
12. Grading

State	Score
RESET	5%
START	25%
FINISH	10%

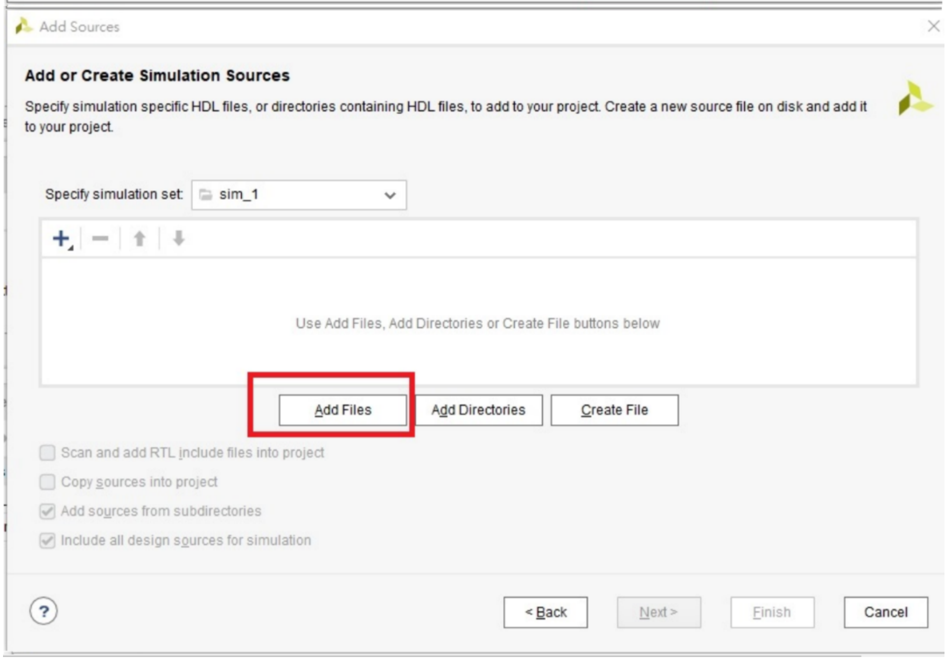
Appendix: How to add pattern.dat to the simulation sources.


Add Sources

Add Sources

This guides you through the process of adding and creating sources for your project

Add or create constraints [ ]
Add or create design sources [ ]
[✓ Add or create simulation sources]


Add Sources

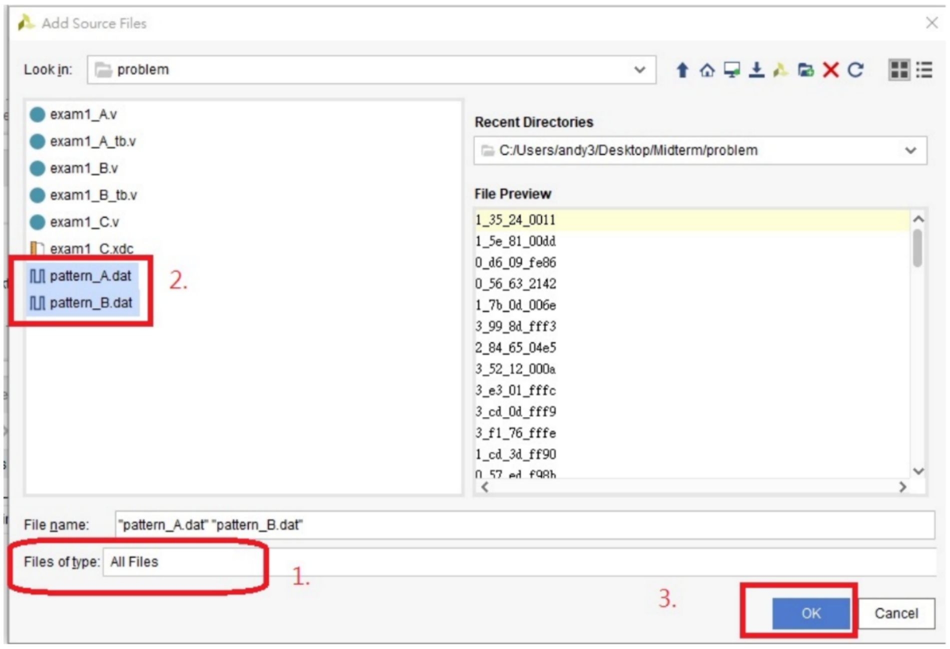
Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Use Add Files, Add Directories or Create File buttons below

Scan and add RTL include files into project [ ]
Copy sources into project [ ]
[✓ Add sources from subdirectories]
[✓ Include all design sources for simulation]



Recent Directories: C:/Users/andy3/Desktop/Midterm/problem

File Preview

```
1_35_24_0011
1_5e_81_00dd
0_d6_09_fe86
0_56_63_2142
1_7b_0d_006e
3_99_8d_fff3
2_84_65_04e5
3_52_12_000a
3_e3_01_fffc
3_cd_0d_fff9
3_f1_76_fffe
1_cd_3d_ff90
0_57_ed_f98b
```

File name: "pattern_A.dat" "pattern_B.dat"

Files of type: All Files

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!!"