Lab 7: Digital Photo Frame

Submission Due Dates:

Demo: 2022/11/29 17:20

Source Code: 2022/11/29 18:30

Report: 2022/12/04 23:59

Objective

1 Getting familiar with finite state machines (FSMs) in Verilog.

2 Getting familiar with controlling the VGA display, block RAM, and other I/Os on the FPGA demo board.

Action Items

In part A of this lab, you will implement a digital photo frame with fancy transition effects on the VGA display. In part B, you will implement a sliding puzzle game with a photo on the VGA display. Pick an image you like. Use **PicTrans.exe** to generate your own .coe file**. Make sure it is an appropriate image**. Template and constraint files are provided for both part A and part B. VGA sample code and **PicTrans.exe** are in the **SampleCode_VGA** folder.

`A. lab7_1.v` (40%)

Design a VGA controller that makes your image scroll up or down over time and mirrors its orientation.

a. IO list:

- Inputs: clk, rst, en, dir, vmir, hmir
- Output: vgaRed, vgaGreen, vgaBlue, hsync, vsync

b. **rst**: the positive-edge-triggered reset, the VGA display will show the image at the original position after triggered.

c. If **en** == 1'b0: hold the image on the screen unchanged.

d. If **en** == 1'b1:

- If **dir**== 1'b0:
  - the image scrolls up at the frequency of 100MHz divided by 2^{22}.



- If **dir**== 1'b1:
  - the image scrolls down at the frequency of 100MHz divided by 2^{22}.



e. If **vmir**==1'b1: the image is mirrored vertically.
f. If **hmir**==1'b1: the image is mirrored horizontally.
g. If both **vmir** and **hmir** ==1'b1, the image is mirrored **both** vertically and horizontally.
h. If both **vmir** and **hmir** ==1'b0, the image remains in its original orientation.
i. When the image is mirrored, the scroll direction should not change.


original


mirrored horizontally


mirrored vertically

Demo video:
https://drive.google.com/file/d/1B-KhZnL6TXjvlks2CgUi70PTjyu3oa4I/view?usp=sharing

j. You have to use the following template for your design:

```
module lab7_1 (
  input clk,
  input rst,
  input en,
  input dir,
  input vmir,
  input hmir,
  output [3:0] vgaRed,
  output [3:0] vgaGreen,
  output [3:0] vgaBlue,
  output hsync,
  output vsync
  );
  // add your design here
endmodule
```

IO Connection:

clk	connected to W5
rst	connected to U18 (BTNC)
en	connected to V17 (SW0)
dir	connected to V16 (SW1)
vmir	connected to W16 (SW2)
hmir	connected to W17 (SW3)
vgaRed	connected to pin N19, J19, H19, G19
vgaGreen	connected to pin D17, G17, H17, J17
vgaBlue	connected to pin J18, K18, L18, N18
hsync	connected to pin P19
vsync	connected to pin R19

B. `lab7_2.v` (60%)

Design a sliding puzzle VGA game. Your image is divided into 4*4 blocks. One of the 16 blocks is empty. Initially, the blocks are misplaced. You can move the blocks with buttons on the FPGA board and re-arrange the blocks back to the original order. This is similar to the well-known "15 puzzle", where you need to re-arrange the blocks so that the numbers are in the correct order. You may try it here in this open-source web game.

a. IO list:

- Inputs: clk, rst, up, down, left, right, hint
- Output: vgaRed, vgaGreen, vgaBlue, hsync, vsync, pass

b. **rst**: The VGA display will show the puzzle at the initial state after the positive-edge-triggered reset. You may set the initial position of each block as you like, but keep in mind that some arrangements are not solvable. You may use the web game mentioned above to get a solvable arrangement.

example of an initial position

example of a solved puzzle

c. **You need to prepare a solution to your puzzle for the TA** to reduce demo time. Or you may use the sample arrangement and solution below:

2	3	4	8
1	6	7	12
5	10		11
9	13	14	15

→

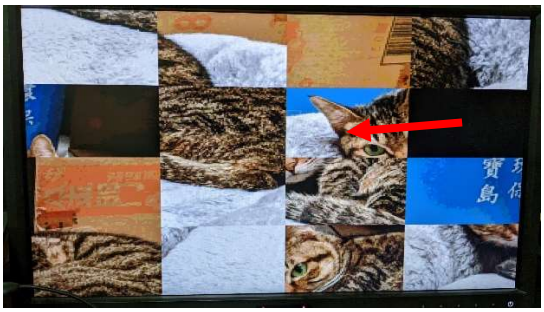
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

The solution for the above puzzle is ←↓↓ →→→ ↑↑↑ ←←←

d. When the **up** button is pressed: The block below the empty block moves up. It does nothing when the empty block is next to the bottom border.


e. When the **down** button is pressed: The block above the empty block moves down. It does nothing when the empty block is next to the top border.


f. When the **left** button is pressed: The block to the right of the empty block moves left. It does nothing when the empty block is next to the right border.

g. When the **right** button is pressed: The block to the left of the empty block moves right. It does nothing when the empty block is next to the left border.


h. If **hint**==1'b1, the monitor shows the correct puzzle solution. The **up**, **down**, **left** and **right** buttons do nothing. That is, the puzzle cannot be moved when **hint**==1'b1.

i. When all the blocks are in the correct position, the **pass** is set to high, and LED0 lights up.

j. If **pass**==1'b1, the **up**, **down**, **left** and **right** buttons do nothing. No blocks can be moved any further until a reset.

Demo video:

https://drive.google.com/file/d/1-RE4OYlrWDOn5ca0ECOnKJaRX2IcsnnG/view?usp=sharing

k. You have to use the following template for your design:

```
module lab7_2 (
  input clk,
  input rst,
  input up,
  input down,
  input left,
  input right,
  input hint,
  output [3:0] vgaRed,
  output [3:0] vgaGreen,
  output [3:0] vgaBlue,
  output hsync,
  output vsync,
  output pass
);
  // add your design here
endmodule
```

IO Connection:

clk	connected to W5
rst	connected to U18 (BTNC)
up	connected to T18 (BTNU)
down	connected to U17 (BTND)
left	connected to W19 (BTNL)
right	connected to T17 (BTNR)
hint	connected to V17 (SW0)
vgaRed	connected to pin N19, J19, H19, G19
vgaGreen	connected to pin D17, G17, H17, J17
vgaBlue	connected to pin J18, K18, L18, N18
hsync	connected to pin P19
vsync	connected to pin R19
pass	connected to U16 (LED0)

Questions and Discussion

Please answer the following questions in your report.

A. If we want to scroll left and right instead of up and down, how would you modify your design of lab7_1?

B. There are 2D arrays in Verilog. Briefly explain how you may use it in your design of lab7_2.

C. The FPGA board we use only has 1800kbits of BRAM, which does not even fit a full resolution (640x480) image. Suppose you want to implement a video game, apart from storing a smaller image (320x240) like we did in this lab, what other method can you use to reduce BRAM usage? (**Give at least 2 methods.**)

Guidelines for the report

Refer to the guidelines in the report template (or in the previous lab assignments).

Grading policy (subject to change): Part (A): **35%**; Part (B): **50%**; Part (C): **10%**; (D): **5%**

Attention

- ✓ DO NOT include the clock_divider, debounce and one-pulse modules in the file you hand in. Please do not integrate them into lab7_1.v and lab7_2.v.
- ✓ If you have two or more modules used for any specific lab, merge them into one

Verilog file before the submission.

- ✓ You should submit two source files, **lab7_1.v** and **lab7_2.v**. DO NOT hand in any compressed ZIP files, which will be considered an incorrect format.
- ✓ You should also hand in your report as **lab7_report_StudentID.pdf** (e.g., lab7_report_110062666.pdf).
- ✓ You should be able to answer the questions for this lab from TA during the demo.
- ✓ You need to prepare the bitstream files before the lab demo to make the demo process smooth.
- ✓ Feel free to ask questions about the specification on the EECLASS forum.