National Tsing Hua University
Department of Electrical Engineering
EE429200 IC Design Laboratory, Fall 2021

Lab 07: Line-based Image Processing (3%)
Assigned on Oct 28, 2021
Due day on Nov 4, 2021

Objective

In this lab, you will learn:

1. Instantiating modules in Verilog design.
2. Basic memory accessing.
3. Advance finite state machine design.

Demo checklist:

- ☐ Draw the state diagram of steganography
- ☐ Draw the state diagram of decoder
- ☐ Show the Spyglass result
- ☐ Show all decoded messages
- ☐ Answer the following questions
    1. What's the DRAM controller behavior?
    2. What's the difference between memory (SRAM and DRAM) and register?

Environment Setup

Copy lab file packages from ee4292. Decompress the package and enter it. You can check the file list in Appendix.

```
$ cp ~ee4292/iclab2021/lab07.zip .
$ unzip lab07.zip
$ cd lab07/
```

Note: please run simulation in the ***sim*** directory to maintain a fine data management.

Description

Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video. In this lab, you are going to find the secret message hidden in the image. As Fig 1. shown, for every 8-bit 3-by-3 block, take the last bit (LSB) of each pixel and it becomes 1-bit 3-by-3 block. The value of the upper left corner of the 1-bit 3-by-3 block is to decide how to traverse the block. After traversing the block, we can get 8-bit data, which corresponds to a particular ASCII

character. After decoding these three rows, continue to read the next three rows and decode them until interpret the whole image.

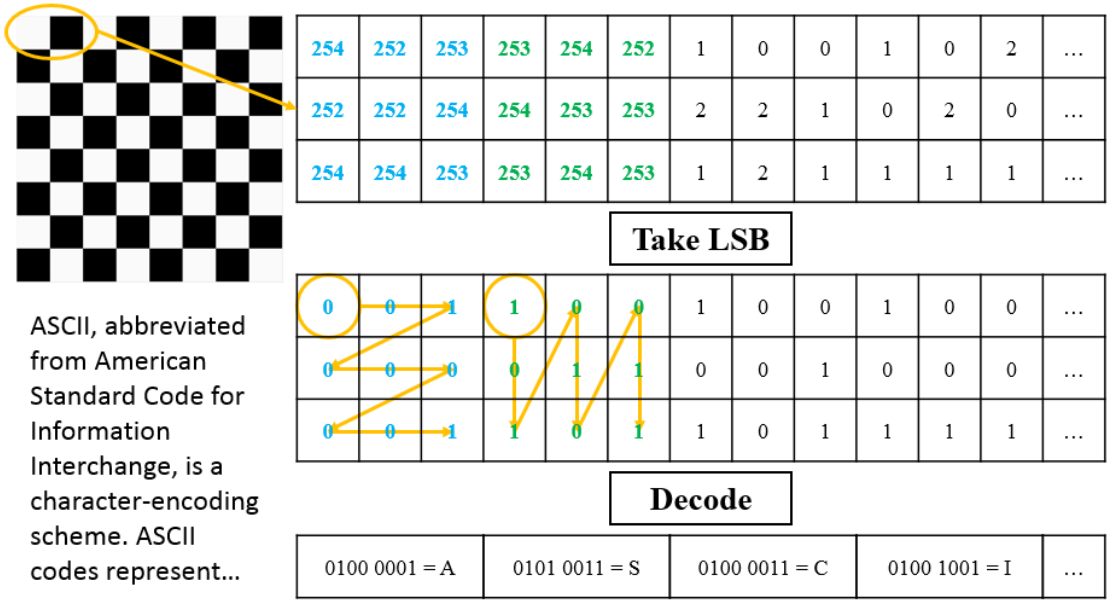

Fig 1. Steganography in this lab.

Fig 2. shows the overall architecture. At first, testbench will clear and load image data to DRAM. Then your design read image data from larger memory (DRAM) and take LSB of it and then write to smaller memory (SRAM). After that, enable the decoder, which read data from SRAM and decode to ASCII code.

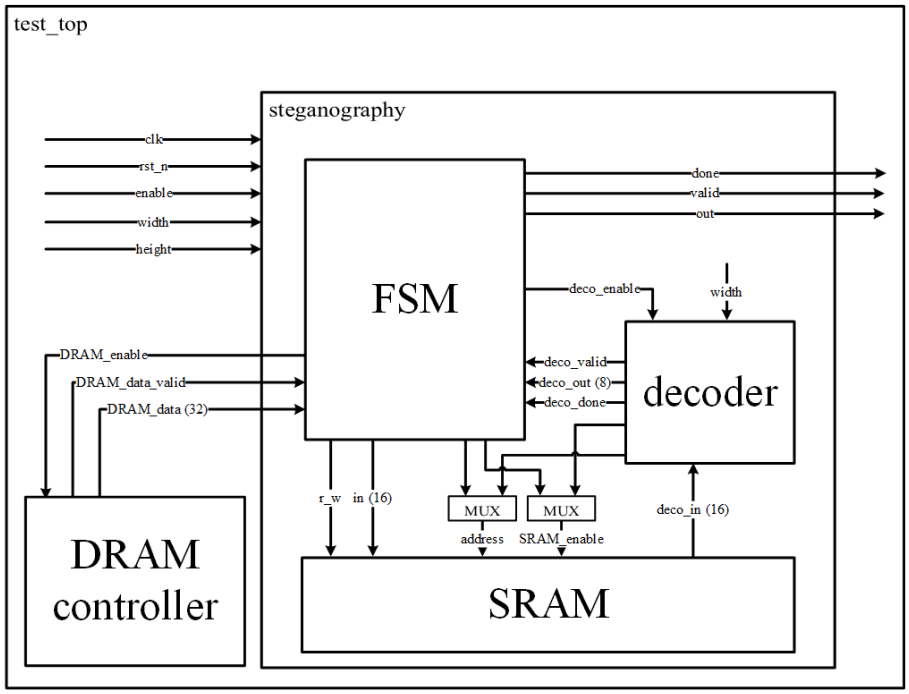

Fig 2. Overall architecture.

Action Items

I. Understand DRAM behavior:

Use **test_dram.v** to understand the behavior of DRAM and DRAM controller in nWave.

1. First, testbench will clear and load image data to DRAM. After enabling the DRAM controller, **DRAMenable** = 1, the data will be ready after several cycles. Once **DRAMdata_valid** turns to logic high, the **DRAMdataout** is avaliable.
2. Fig 3. shows the image is stored in the DRAM in raster scan order (z-scan). Each pixel in an image is an 8-bit data.

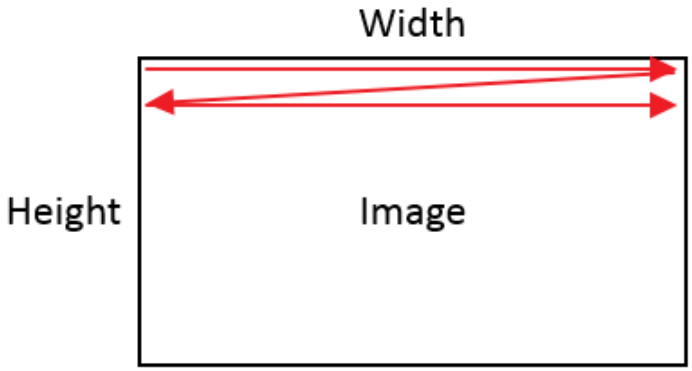

Fig 3. Raster scan order in DRAM.

	col. 0	**col. 1**	**col. 2**	**…**
row 0	pixel 0	pixel 1	pixel 2	…
row 1	pixel W	pixel W+1	pixel W+2	…
row 2	pixel 2*W	pixel 2*W+1	pixel 2*W+2	…
…	…	…	…	…

3. The word width of DRAM is 32-bit but it only stores 3 pixels at each address. The byte order is Little-Endian.

addr 0	0	pixel 2	pixel 1	pixel 0
addr 1	0	pixel 5	pixel 4	pixel 3
addr 2	0	pixel 8	pixel 7	pixel 6
addr 3	0	pixel 11	pixel 10	pixel 9
…	…	…	…	…

II. Access DRAM and write SRAM.

Implement the control circuit (FSM) in steganography.v to access DRAM and write SRAM.

1. The word width of SRAM is 4-bit, and the SRAM has 128 words in total. Only the LSB of each pixel is useful to find the secret message. But the memory space is not enough to store the whole LSB of image, so store at

most 3 rows each time. We can read DRAM 1 times and take LSB to get 3-bit data and then write SRAM. Store the data come from first row of image at SRAM address 0 to 39, second row at address 40 to 79, and third row at address 80 to 119.

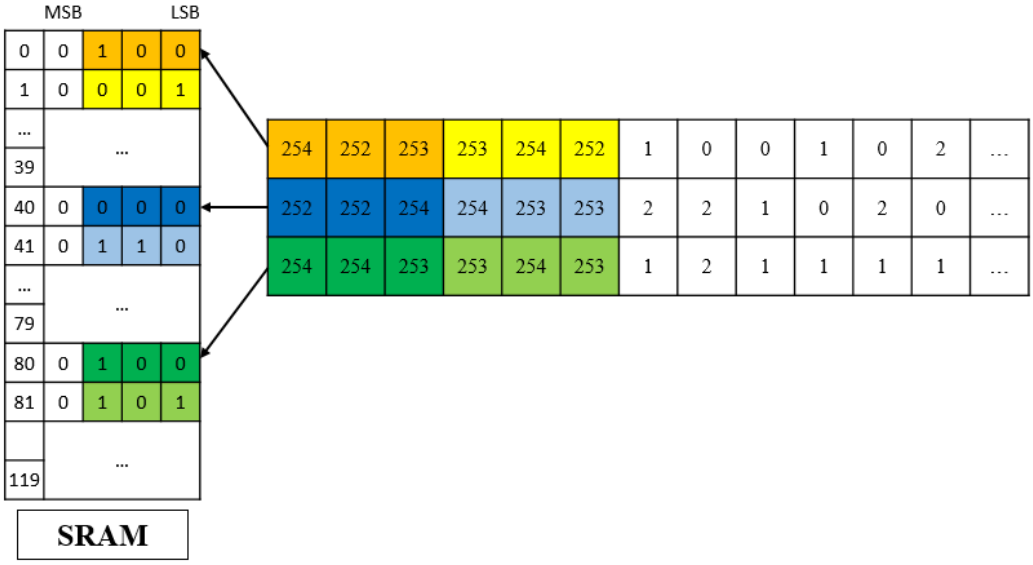

Fig 4.

2. Check whether your data is stored in the SRAM successfully or not. The testbench testsram.v may be helpful for you to check the SRAM. a. After reading the first 3 row of the image, set **SRAMenable**=1 and **r_w**=1. Or understand how the testbench works, write your own testbench to test your FSM, and check LSB in nWave. b. The file *chess.txt* shows all the LSB for each pixel.

III. Access SRAM and decode messages.

Implement the decoder in decoder.v to access SRAM and decode messages.

1. Read 3 rows of SRAM to decode the hidden messages. Access address 0, 40, 80 and store the data in buffer, so we have the first block then we can decode the first ASCII code. After that, access address 1, 41, 81 and decode code, until finishing whole blocks.
2. Use test_top.v to check decoder. Modify the test_top.f to test the other two patterns (Change CASE1 to CASE2 or CASE3).

Note that:

(1) Width and height of image are multiple of 12 and smaller than 128.

(2) In nWave, choose signals, then press **Waveform > Signal Value Radix > ASCII**. You can see ASCII character in nWave.

Appendix

Directory	Filename	Description
hdl	steganography.v	Your design here
hdl	decoder.v	Your decoder here
hdl	dram_read_controller.v	DRAM controller
hdl	SRAM.v	SRAM module
hdl	two_port_ram_sclk.v	DRAM module
sim	chess.txt	LSB of image chess_48x48_en.yuv
sim	simple_task.v	Some tasks
sim	test_dram.f	Test filelist and Testbench
sim	test_dram.v	Test filelist and Testbench
sim	test_SRAM.f	Test filelist and Testbench
sim	test_SRAM.v	Test filelist and Testbench
sim	test_top.f	Test filelist and Testbench
sim	test_top.v	Test filelist and Testbench
sim	Image/	Image and hidden messages