CSC B58 - Lab 7

Memory and RGB Video

1 Learning Objectives

The purpose of this lab is to learn how to create and use Random Access Memories (RAMs) as well as simple animations using the built-in RGB Video in Logisim.

2 Marking Scheme

This lab worth 4% of your final grade, and also includes a bonus part that worth 1% of your final grade. You will be graded out of 8 marks for this lab, as follows:

- Prelab (including Simulations, Schematics and State Diagram): 3 marks
- Part I : 2 marks
- Part II : 3 marks
- Part III (Bonus): 2 marks (1 mark for prelab, 1 mark for inlab; 1% of your final grade)

Preparation Before the Lab

You are required to complete Parts I and II of the lab by building and testing your circuit in Logisim. You should include your state diagrams, schematics, your circuit design in Logisim, and simulation outputs for Parts I and II in the prelab.

You are required to implement and test all of Parts I and II of the lab and demonstrate them to the TAs.

3 Part I

In addition to logic blocks and flip-flops, contemporary FPGA devices provide flexible embedded memory blocks where you can configure parameters like the number of memory locations and the number of bits at each location. In this part of the lab, you will create a small RAM block and fill it with values. In Logisim, you can find the RAM unit under *Memory > RAM*.

A general sketch of a memory module is shown in Figure 1. It consists of a memory block, an address register, a data register and a control register (we're only looking at write operations in this example). You can see that the address, input data, and the write enable control signal are all stored in registers before being presented to the memory unit. The registers are used here to model the address and data registers on a typical processor and serve the function of keeping the input values stable for a full clock cycle while the inputs are being changed between clock cycles.

As seen in Figure 2, the RAM module in Logisim has these ports:

1. *A*: On the top left, the 5-bit input *A* is *Address* in Figure 1.
2. *M3 [Write enable]*: Just below *A*, M3 is *WriteEn* in Figure 1.
3. *M2 [Output enable]*: this signal is not shown in Figure 1 but is used for reading values. If *output enable* is 1, the the memory at the address will be displayed on *DataOut*.
4. *C1*: The clock for this memory unit.
5. On the left and right side of the memory block are *DataIn* and *DataOut*.

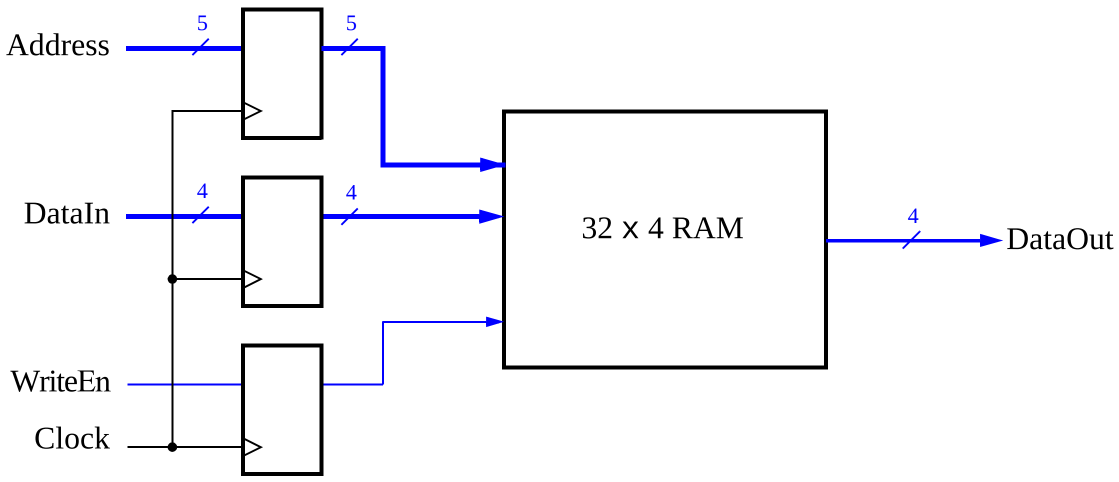

Figure 1: Schematic of the 32 × 4 embedded memory module.

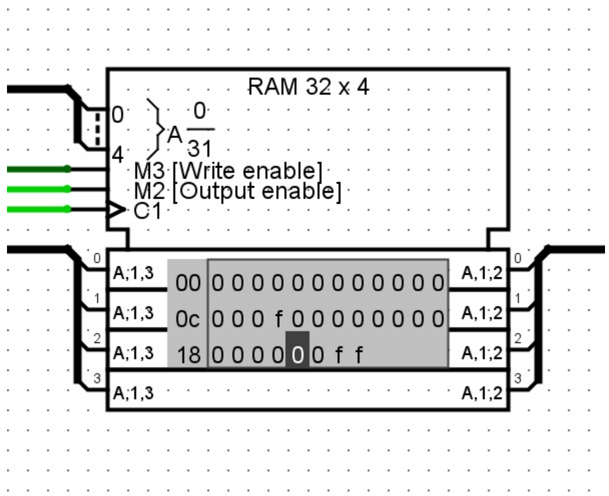

Figure 2: RAM component in Logisim

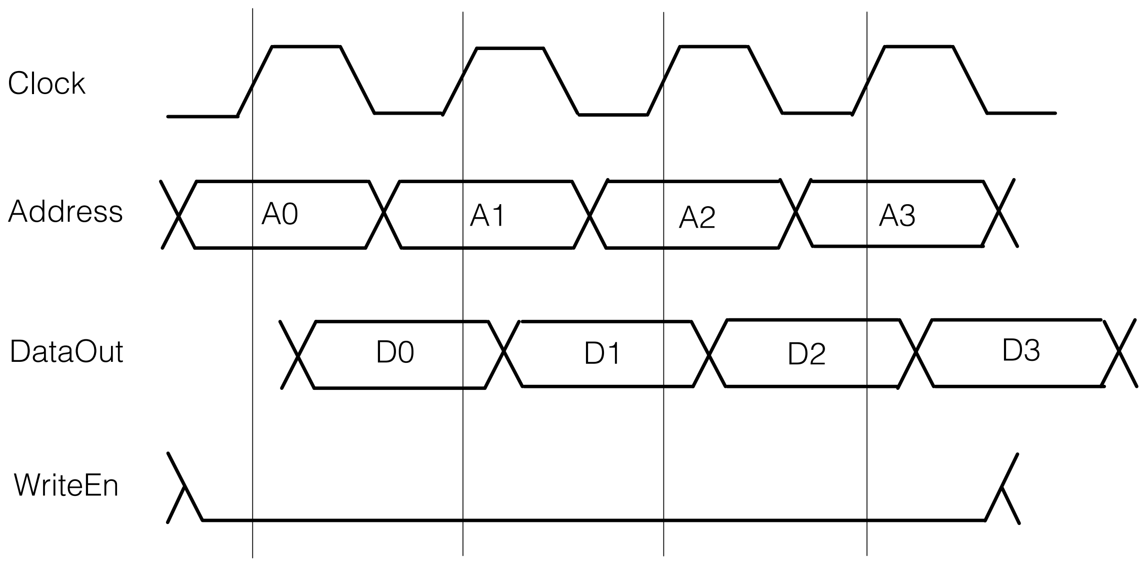

Figure 3: Timing diagram for read operations.

A timing diagram showing reading of the memory is shown in Figure 3. Four locations at addresses *A0*, *A1*, *A2* and *A3* are accessed and the corresponding data *D0*, *D1*, *D2* and *D3* are read from those addresses, respectively. Figure 4 shows the timing for writing data to the memory. Observe that *WriteEn* is only high for addresses *A1* and *A2*. This means that only data words *D1* and *D2* are written, respectively.

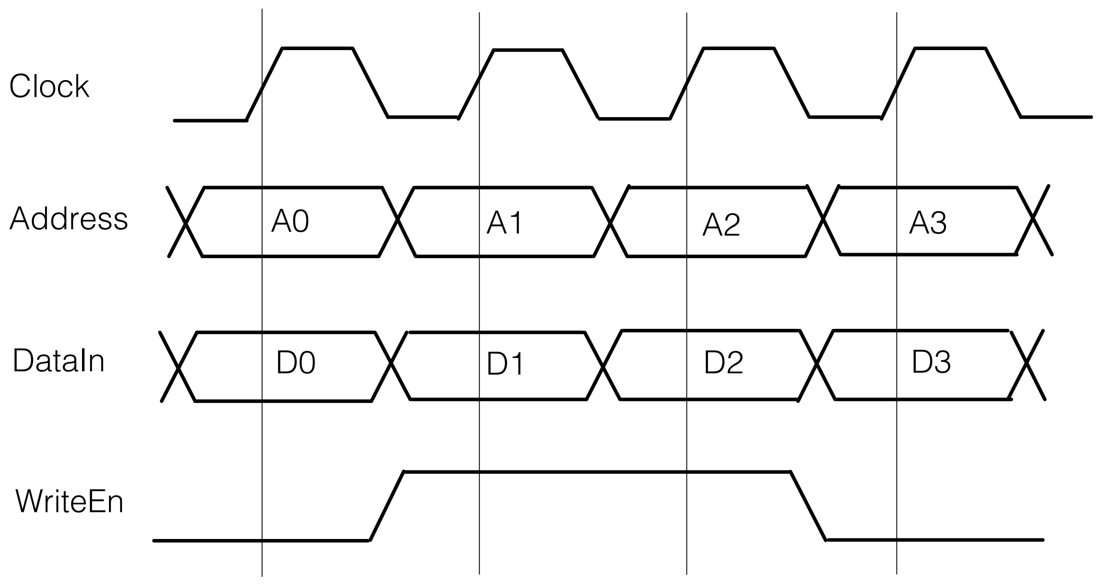

Figure 4: Timing diagram for write operations. Note that only addresses *A1* and *A2* are written.

Perform the following steps:

1. Create your 32 × 4 RAM module in Logisim. You can find the component under *Memory* in components. Then you should adjust the address bit width and data bit width width in *Properties*. **(PRELAB)**
2. Notice that the RAM module in Logisim has both *Write Enable* (for writing to memory) and *Output Enable* (for reading from memory). What happens if both signals are off when the clock goes high? What happens when both signals are on? Experiment with these signals to confirm your guess and write the behaviour in your prelab report. **(PRELAB)**
3. Create circuitry around this RAM unit that fills each location in memory with increasing values, starting with a zero value at memory location 00000, a value of 1 at memory location 00001, and so on. Once you get to memory location 10000, start again with a value of zero and keep writing increasing values until you write the value *F* at memory location 11111.
4. Connect the output of the memory unit to a seven-segment display.
5. Draw a schematic describing this circuit as part of your preparation. **(PRELAB)**
6. Test your modules with *Poke*(👆) and the Logisim clock to verify its correctness. Include a few screenshots that shows the contents of the memory unit during your simulation. **(PRELAB)**

4 Part II

For this part you will learn how to display simple images on the RGB Video display in Logisim. Your task is to design a circuit to draw a *filled* square on the screen at a specified location. This is a model of a typical VGA screen like the ones used in the DE1-SOC labs.

The first step is to put a RGB Video component on your canvas. The RGB Video component can be found under *Input/Output* in the component listing. We will use the default size of a 128 × 128 RGB Video. If you wanted to change the size of your RGB Video, click on it to view its properties, then change the values in the *Matrix Columns* and *Matrix Rows*.

Background

The RGB Video component has 6 inputs along the bottom edge (listed here from left to right):

1. *Reset*: Set this to 1 to reset the screen (makes the screen black)
2. *Clock*: Pixel changes occur on the rising clock edge
3. *Write Enable*: Write the specified colour into the pixel at the given X and Y coordinate.
4. *X Coordinate*: A 7-digit input used to specify the horizontal position of the pixel to write.
   - An input value of 0000000 indicates the left-most column, 1111111 (127, width-1) indicates the right-most column (width-1)
5. *Y Coordinate*: A 7-digit input used to specify the vertical position of the pixel to write.
   - An input value of 0000000 indicates the top row, 1111111 (127, height-1) indicates the bottom row
6. *Data In 888 RGB (24 bit) format*: This 24-bit value is actually three 8-bit values concatenated together, where the first 8 bits specify the red component, the next 8 bits specify the green component and the last 8 bits specify the blue component of the pixel to write.
   - If all three of these 8-bit values are set to zero, the resulting pixel is black. If all three are set to 11111111 the pixel is white.
   - If you're unfamiliar with how colours combine in an additive way for light (as opposed to the subtractive way for paint), you should look this up and learn how to make colours like magenta, cyan and yellow out of red, green and blue.

Drawing Squares on the RGB Video - Expected Behaviour

Your circuit will accept an X position and Y position as input. Both X and Y values should be 7 bits long for a 128×128 RGB display.

Once the Enable signal is turned on, the circuit should draw a square whose size is 16×16 pixels, and whose *top-left* corner is at the (X position,Y position) specified by the input. The square should be filled with the colour of your choice, as long as it's not white or black. If you really want to impress your TA, have multiple colours in a single square (!!).

After a square has been drawn, your circuit should allow additional squares to be added to the RGB Video display at different locations. This would be done by turning the Enable signal off before updating the values of X and Y before turning Enable back on (like lifting a pen and putting it down again).

In your prelab report, answer the following questions: **(PRELAB)**

1. What happens if you don't turn Enable off before updating X and Y?
2. What happens if you turn Enable off before 256 clock cycles have passed?
3. What happens if you turn Reset on while Enable is on?

Note: if the X and Y coordinates are less than 16 pixels away from the bottom or right edge of the display, have the square wrap around to the top or left side of the display when drawing it.

The high-level design of the circuit for the system is given in Figure 5. It contains 3 major blocks:

1. The RGB Video is responsible for the drawing of pixels on the screen. It requires the correct X and Y inputs, the colour values, the Write Enable signal and the clock and reset.
2. The datapath that contains arithmetic circuitry and registers and is controlled by the FSM to produce the RGB Video input signals needed to draw the 16×16 filled square (other than clock and reset).

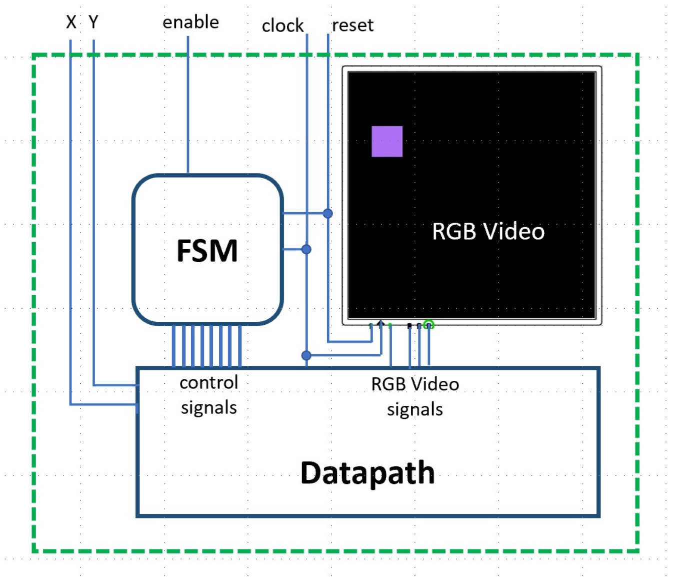

Figure 5: Design Overview - State Machine, Datapath and RGB Video.

3. A finite state machine that serves as a controller for the datapath. The datapath control signals are illustrated in Figure 5, however the number of control signals and what they control will depend on the structure of your datapath.

To complete this part of the lab, perform the following steps:

1. Draw the schematic of a datapath that implements the required functionality and build it in Logisim. Simulate the datapath to confirm that it sends the correct output values, given the datapath control signals that you provide. Include schematics, Logisim file, and screenshots of simulation output in your prelab. **(PRELAB)**
2. Design the FSM that sends the necessary signals to the datapath and then implement that FSM in Logisim. Your simulation for this part should only include the FSM. **(PRELAB)**
3. Connect your FSM, datapath and the RGB Video together. You should simulate the combination of them using *Poke*(👆) before connecting your circuit to the clock. Include screenshots of your simulation. **(PRELAB)**
4. Demonstrate your design to the TA. **(IN-LAB)**

Advice

Before you design your datapath or FSM, experiment with the RGB Video component so that you're comfortable with how it works. Try drawing a single pixel on the screen to ensure that you understand how the RGB Video works.

You should start with a slow clock speed so that you can observe how your circuit is drawing the pixels onto the display. Once you're confident about your design, speed up the clock and try drawing multiple boxes.

Since this is the last lab, the design of the FSM and datapath are left to your discretion. You may use any components from previous labs or from the component library in Logisim. Your FSM might require several states or only a few. Your datapath might use several components or not. It's all up to you at this point.

This being said, we'll go over some ideas in tutorial that might help you plot the 256 pixels of the 16×16 color-filled square.

5 Part III - Bonus

In this next part we will expand on Part II of this lab by animating the box and having it bounce around the screen.

The (X,Y) location of the box (still 16×16 pixels) will be controlled by your circuit and will change over time.

To accomplish the animation, your circuit will have to make it seem as though the box is seamlessly moving around the screen. It will do this by erasing and redrawing the box each time it is to be moved. We would like the box to always move in a diagonal fashion (one pixel vertically and one pixel horizontally), four times per second.

You will implement the circuit in two steps. First, you will design the datapath for a module that is able to draw (or erase) the image at a given location. The datapath of this circuit will be very similar to the circuit used for Part II. One addition to your circuit will be two counters: one for the current location of X and the other for the current location of Y. You will also need a way to store the current direction (horizontal and vertical) of the box.

The (X,Y) counters will be able to count up or count down since the box can be moving in any direction on the screen. For this part though, the box should not wrap across the bottom or right edge of the screen like in Part II. The box should "bounce" just before that happens.

To implement this *bounce* off the edges of the screen, the current location of the box and direction of travel should be used to update the direction registers. For example, if the box is moving in the down-right direction and the next position of the box would move it off the bottom of the screen, the vertical direction bit would be flipped indicating the box should start moving in an up-right direction. Likewise, if the box was moving in the down-right direction and the next position of the box was further than the right edge of the screen, the horizontal direction bit would be flipped indicating the box should start moving in a down-left direction.

A block diagram of your circuit is shown in Figure 6. It is not complete and lacks some details and signals. Consider it only as a starting point. Note the presence of the Delay Counter here, which is needed when you have a fast clock, to make sure your animation runs slow enough to be observable.

Use the same switches as you used in Part II, as needed. Remember that X and Y inputs of that module are no longer input from the switches.

A rough outline of the algorithm is as follows:

1. Use the Part II datapath to draw the box in the current location and then erase the current box.
2. Update Counter_X, Counter_Y based on the direction registers. Update the direction registers themselves (if necessary).
3. Go to Step 1.

Implement the circuit by completing the following steps (required if you are attempting the bonus part):

1. Design (draw the schematic and build in Logisim) and simulate a datapath that implements the required functionality. **(PRELAB)**
2. Design (draw state diagram and build in Logisim) and simulate an FSM that controls the implemented datapath. **(PRELAB)**
3. Put the FSM, datapath and the RGB Video together. Test your circuit thoroughly and demonstrate the working circuit to your TA. **(IN-LAB)**

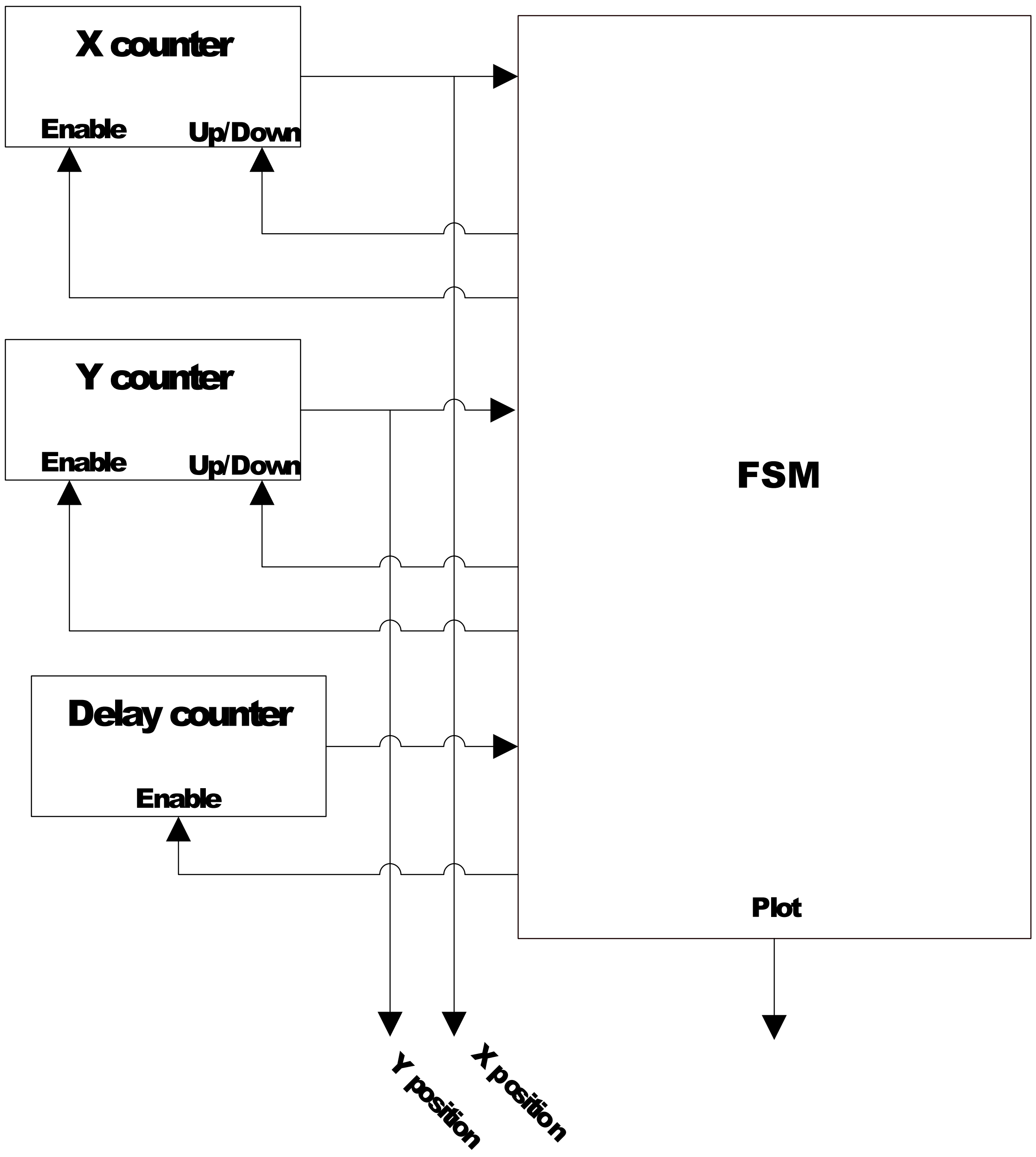

Figure 6: Rough schematic for your animated image circuit. There may be signals and pieces missing.