Laboratory Exercise 8

Animation on the VGA Display

Revision of November 11, 2022

The purpose of this exercise is to use what you learned in Lab 7 to draw boxes on the VGA display and develop an animation of a bouncing box on the VGA display. By doing this, you will learn the basic principles of how animation works at the level of what was done in the original *Pong* video game. See `https://en.wikipedia.org/wiki/Pong` to learn more about *Pong*. According to that Wikipedia page, *Pong* was first released 50 years ago in 1972 and started the video game industry so 2022 is an interesting anniversary for video games.

1 Workflow

For each part of the lab, you should begin by writing and testing Verilog code, using ModelSim. Once your design works in simulation, you should compile it with Quartus. You must simulate your circuit with ModelSim using reasonable test vectors written in the format used in Lab 2 for the simulation files. You should be prepared to show schematics, Verilog, and simulations to your TA, if requested. You will not be helped with debugging your code if you do not have schematic and ModelSim simulations. Issues identified with the tester, without doing your own ModelSim simulations will not get help.

Warning: This lab document is quite detailed and covers a lot of new information. Please read through it carefully. Do not rush to just get the labs done as you will spend much more time debugging your code than making progress. Also, it is highly recommended that you watch the tutorial videos on Quercus in the Lab 7 page.

2 Part I

In this lab, we will create a simple animation of the box from Lab 7 Part II by having it bounce around the screen.

The skeleton code for the top-level module of your design, called `part1`, is provided to you in the file `part1_template.v`. Copy this file and rename it to `part1.v`. This is the file you will submit to the Automarker after you add your code.

Note that the template includes three parameters, `X_SCREEN_PIXELS` and `Y_SCREEN_PIXELS`, which define the size of screen being used and `CLOCKS_PER_SECOND`, which specifies the clock

frequency. The default frequency is set to 5 KHz. As in Lab 7 Part II, the pixel dimensions specify the screen size. The Automarker will change the parameters to test a 9×7 screen. The Automarker will also set `CLOCKS_PER_SECOND` to 1200. Again, these parameters should be used in your design and will be helpful for scaling the design for simulation. For example, you should use `CLOCKS_PER_SECOND` when initializing a counter that might generate a pulse every second. For simulation, this parameter can be set to a small number, but when running on the board with a 50 MHz clock, then instantiate the module with the parameter set to 50 million. Some other parameters are defined and given as examples of how you can use parameters as part of other parameters. You may use these if you find them useful, but they are not required by the Automarker.

The colour of the box (4×4 pixels) will be selected by the input `iColour` but now the `(X,Y)` location of the box will be controlled by your circuit and will change over time. Since your circuit is providing the `X` value, you now can also generate all eight bits so that you can access the full screen in this part.

The module also has `iResetn` and `iClock` inputs used in the same way as in Lab 7 Part II. The outputs of the module are also used in the same way as in Lab 7 Part II.

You will need an additional output:

`oNewFrame` This must be set high for exactly 1 clock cycle when your circuit has completed drawing/modifying a frame. With the given VGA and CLOCK parameters in the Automarker, the expected number of cycles between `oNewFrame` going high (i.e., clock cycles per frame) is 20. Given the difficulty of achieving this specification, your circuit can use up to 30 cycles per frame. Going above 30 cycles will result in failure of the test cases as the Automarker cannot wait on `oNewFrame` going high indefinitely. However, if your circuit can achieve less than 24 cycles per frame, you will be given bonus marks up to 20%, 5% for each frame less. E.g., if you achieve 20 cycles per frame you'll be given a bonus of 20%, 15% for 21 cycles, 10% for 22 cycles, and 5% for 23 cycles.

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. Assuming the background starts out black, then erasing is simply to draw black pixels over top of the region being erased. An incremental step would be to just move the box without erasing it. This will draw all over your screen, but is easier because you do not have to figure out erasing.

2.1 VGA Update Rate

The actual VGA adapter updates the monitor at 60 Hz or 60 frames-per-second (fps), meaning that the entire contents of the frame buffer are output to the monitor every 1/60th of a

second. Your circuit should only erase and redraw the box no faster than this rate.

If you have ever looked at the specifications for a monitor that you buy, you will see common rates of 30 fps or 60 fps. In a more sophisticated VGA adapter, there are actually two frame buffers: one is being updated with the image for the next frame, while the other is being output to the monitor. This is called double-buffering and allows a much smoother transition from one frame to another. In our case, there is only one buffer and if you look really carefully you may see a moving image *tearing*, meaning that while you are redrawing it in the new position, part of it is displayed in the old position and part of it is in the updated position.

2.2 How to think about this design

We would like the box to always move in a diagonal fashion at four pixels per second.

You should use a counter, called `DelayCounter`, to track how much time has passed. The `DelayCounter` should generate an enable pulse every 1/60th of a second, which corresponds to the period at which the VGA adapter updates a frame. You should also use a second counter, called `FrameCounter`, to track how many frame times have elapsed. If we want the box to move at four pixels per second, the box should only move one pixel every 15 frames.

Clock Rates Remember that if you are using a physical DE1-SoC board, `Clock_50` runs at 50 MHz. The Automarker will set the parameter `CLOCKS_PER_SECOND` to 1200. Keep these in mind when building the `DelayCounter`. The `DelayCounter` is at least one place where you should use the `CLOCKS_PER_SECOND` parameter that can be changed at the time the `part1` module is instantiated.

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 basically be the circuit used for Lab 7 Part II. In addition, you will need two counters, `CounterX` and `CounterY` that will contain the current (X,Y) location of the box as well as two single-bit direction registers, `DirH` for horizontal and `DirY` for vertical, that will track the direction the box is moving. The `CounterX` and `CounterY` counters will be able to count up or count down since the box can be moving in any direction on the screen. The two direction registers will track the current diagonal direction of the box (up-left, up-right, down-left, down-right). To implement the *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

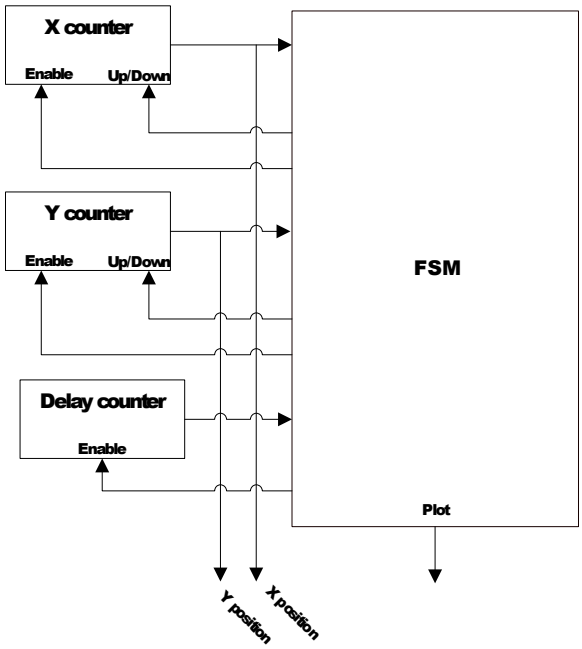

Figure 1: Rough schematic for your animated image circuit. There may be signals and pieces missing.

right edge of the screen, the horizontal direction bit would be flipped indicating the box should start moving in a down-left direction.

A rough schematic of your circuit is shown in Figure 1. It is not complete and most likely lacks some pieces and some signals. Consider it only as a starting point.

A rough outline of the algorithm is as follows:

1. Reset `DelayCounter` and `FrameCounter` to 0. Reset `CounterX` and `CounterY` to 0. Reset `DirH` to 1 and `DirY` 0 to indicate down-right. By doing this, the box always starts moving from the upper-left corner of the screen and moves in a down-right direction.
2. Use the Lab 7 Part II datapath to draw the box in the current location.
3. Reset `DelayCounter` and `FrameCounter` to 0. Then count 15 frames.
4. Use your Lab 7 Part II datapath to erase the current box.
5. Update `CounterX` and `CounterY` based on the direction registers. Update the direction registers themselves (if necessary).
6. Go to Step 2.

Implement the circuit by completing the following steps.

1. Design (draw the schematic and write Verilog) and simulate a datapath that implements the required functionality.

2. Design (draw state diagram and write Verilog) and simulate an FSM that controls the implemented datapath.

3. Do your simulations with Modelsim, again without the VGA adapter. Aside from `iClock`, the only inputs you can set in your simulation are `iColour` and `iResetn`. Check that the outputs are behaving correctly. Once you are satisfied that your circuit is behaving correctly, you can submit it to the Automarker.

4. You can now proceed to test your design on the real hardware.

5. Use the same switches and pushbutton that you used in Part II for `iColour` and `iReset`. They are shown again in Table 1.

`part1` Port Name	Direction	DE1-SoC Pin Name
`iClock`	Input	Clock_50
`iResetn`	Input	KEY[0]
`iColour[2:0]`	Input	SW[9:7]

Table 1: Module `part1` mapping to DE1-SoC pin names

   Simulate your new circuit with the DE1-SoC connections with ModelSim to ensure that you have done the connections correctly.

6. Now you can build your top-level design with the VGA adapter as you did in Lab 7 Part II.

7. Compile the project to generate a bitstream to make sure your code can at least be synthesized.

8. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit by toggling the various inputs and observing the outputs.

3 A Note About Simulating with Quartus Libraries and the VGA Adapter

If you are adding IP blocks (functional blocks of Verilog) generated from Quartus, such as RAMs, you will need to include the simulation libraries for those IP blocks. The VGA adapter, and other modules you might use for projects using the DE1-SoC board, like a

keyboard controller, may also include Quartus IP blocks. You will need to include some Altera simulation libraries (ex., `lpm_ver, altera_mf_ver`) to run your simulations.

The following shows how to add a library in your Modelsim scripts:

```
vsim -L altera_mf_ver <module-name>
```

If you need to add several libraries, then you will need additional `-L library_name` flags.

4 Submission

Please submit `part1.v`. For module names and port names, please follow the provided template.