ECE 212 – Digital Circuits II

Lab 2 – RGB LED Matrix

February 1, 2019

1. Introduction

In this lab you will design and test a circuit to control an Adafruit RGB LED Matrix display and to show a fixed pattern on the display. In Lab 3 you will expand this basic function to create a scrolling display that allows the contents to be change based on user input.

Together with Lab 3 this will provide a challenging design experience in which you create a moderately complex system using fully synchronous design techniques, including the enable circuit you worked with in Lab 1.

2. Background

2.1 The LED Matrix

The LED Matrix used in this lab contains 512 RGB LEDs arranged in a 16 x32 matrix. Each LED is similar to the RGB LED on the Nexys4 FPGA board that we used in ECE 211 last semester in that it contains separate red, green, and blue LEDs which can be turned on independently to display different colors. When combined into a two-dimensional array each RGB LED functions as a "pixel" (picture element) just as in a standard video display. For this reason, in the following we will refer to individual RGB LEDs as pixels.

While with a sufficiently complex control circuit each pixel can display virtually "any" color, for this lab you will create a controller that allows each pixel to take on one of the 8 basic color combinations of red, green, and blue (red, green, blue, cyan, magenta, yellow, white, and black).

The LED matrix is controlled using a fairly complicated interface that uses only twelve signals to control the 512 pixel LEDs. This interface is described in detail in the attached "**LED Matrix Overview"**. Please read over this document before proceeding further in this handout. Prof. Watkins also created a YouTube video describing this interface that you may find useful. See https://www.youtube.com/watch?v=qwImSaRfcJM.

When reading the LED Matrix Overview you will notice that the design of the driving logic on the LED Matrix violates two key guidelines of fully synchronous sequential design: it uses latches to store the row pixel values, and it uses a clock signal that is non-continuous. However, we must work with this device as it is while following these best practices in our own hardware designs.

Another observation about the LED Matrix interface is that it is has fairly stringent timing constraints. A microcontroller-based design is unlikely to be fast enough to meet these constraints. In this application custom logic designed in an FPGA will be much more effective.

2.2 The Design Task

The goal of this lab is to create a controller circuit for the LED Matrix. This controller has two tasks: (1) generate the timing signals required for the LED Matrix interface; and

(2) provide pixel data to create some kind of useful display. We can implement these two tasks in separate modules as shown in Figure 1.

The *sequencer* circuit generates the timing signals for the LED matrix. As described in the LED Matrix Overview document, it needs to generate the `sclk`, `lat`, and `blank` signals along with a `disp_row` signal which enables the current content of the row latch for display. At the same time, it indicates to the pixel generator which pixel values should be output on the `rgb1` and `rgb2` ports by providing the locations of the pixel values that must be provided as inputs to the two shift registers in the display using the `row` and `col` output ports. As described in the LED Matrix overview, pixel values for a row must be provided while at the same time the pixels from the previous row are being displayed; for this reason, `disp_row` must be equal to `row-1` (modulo 8).

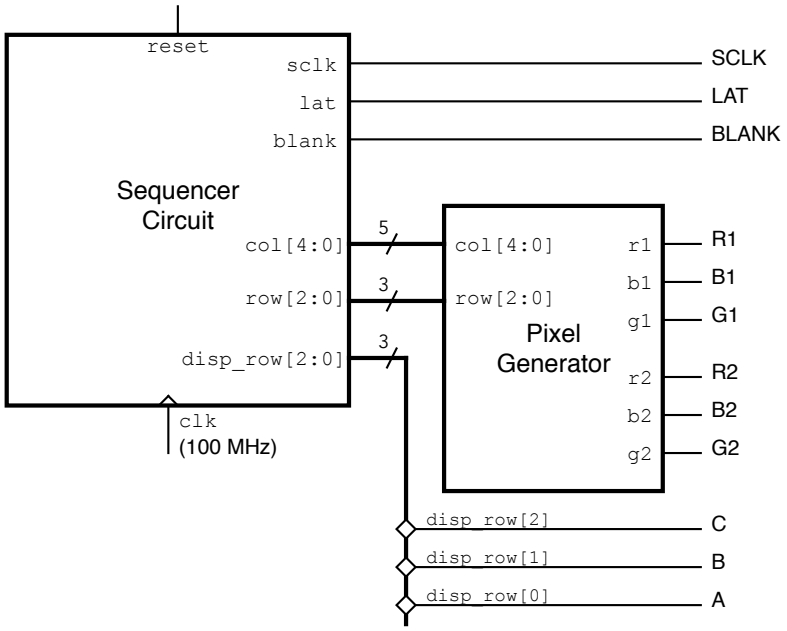

Figure 1 – LED Matrix Controller Organization

The pixel generator circuit uses the `row` and `col` values to produce the two pixel values that are needed for the upper and lower half-panels of the LED Matrix. It can be implemented in a number of different ways depending on what is to be displayed. For example, video displays often use a memory called a *frame buffer* that can stores values for each pixel in the display. We will use this approach next week in Lab 3. However, for abstract patterns we can use combinational logic to create outputs by "turning on" certain pixels when the `row` and `col` values fall within particular address ranges. We will use this approach in this week's lab to debug the sequencer design before adding the complexity of a frame buffer.

You will notice that in this assignment and the provided documentation there are many unspecified details. This is typical in the design of "real" systems; it is the job of the design engineer to work out these details in a way that renders and effective design.

Your design should meet the following requirements:

- The sequencer and pixel generator must be implemented as separate modules that are instantiated in a top-level module for implementation on the FPGA. You will need to reuse the sequencer design in next week's lab, so you need to be able to cleanly separate it from the pixel generator.
- The finished design should display pixels without showing any noticeable flicker. This implies an overall refresh rate that "redraws" every pixel in the LED matrix at least 100 times a second.
- To ensure proper operation of the LED Matrix, all control signals must be stable for at least 300 ns.
- You may implement the pixel generator to display any pattern that meets the following requirements:
    - All eight colors are displayed at least once in both the top and bottom half of the array,
    - At least one pixel in the far left column must be on and have a different color than the pixel in the 2nd column of that row.

  Figure 2 shows an example pattern that meets these requirements. However, you need not duplicate this exact pattern – be creative!
- Your overall design must follow good design practice for sequential design and SystemVerilog coding – refer to the coding guidelines discussed in class. Some of the more important requirements:
    - Your design should run off the 100 MHz clock provided on the Nexys4 board. All flip-flops in your design should be connected directly to this clock. To create lower-rate signals, use the "`delay_enable`" or "`rate_enable`" module provided in class or create your own counter-based circuits.
    - The sequencer circuit should have a master reset input that resets all flip-flops to known states.
    - Your design must contain **no latches**. You must document this in your lab report by providing a copy of the Vivado Synthesis report indicating that no latches are present in our design.

For 10% extra credit: parameterize your design to control two or more daisy-chained units.

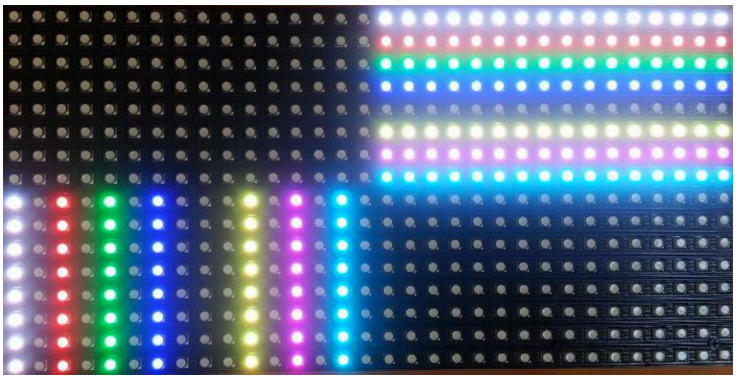

Figure 2 – Example Test Pattern

3. Prelab

This is a complex design that will require a substantial amount of thought and planning if you are to complete it during the lab period. For this reason the prelab assignment for this lab is to create a **design plan** for your LED Matrix Controller. A design plan is a document that describes the planned organization and operation of your intended circuit. It should be clear enough that you could give it to an engineer familiar with digital design and they would be able to implement your design without further information. Required components of this design plan are:

1. **Block diagrams** of your controller circuit and important submodules (particularly the sequencer).
2. **State transition diagrams** for any and all finite state machines in your design. Make sure to show all of the inputs and output of the FSM.
3. **Module descriptions** – a listing of each module in your design by name, followed by at least a sentence or two describing its functionality.
4. **Timing analysis** – In this section you should describe how you will make sure that each controller signal is held stable for at least 300ns and the amount of time you plan for the controller to display each row and why you selected this value. Remember that there are 8 rows to sequence through and each row should be refreshed at least 100 times per second.

Diagrams may be either created with a drawing tool or *neatly* hand-drawn but must be integrated with the rest of the design plan. The end result should be a document in PDF format that is uploaded to Moodle *prior* to the beginning of lab. One design plan is required for each lab group.

4. In the Lab

This section describes the procedure you will follow in the lab. Please read over this procedure before you come to lab; feel free to ask questions at any time about anything you don't understand.

1. Log in to the PC at your lab station and visit the `ece_212_labs` directory of your Git repository. Make sure that your local repository is up to date with respect to GitLab.
2. Create a subdirectory within the `ece_212_labs` directory named `Lab02`. Create subdirectories `RTL`, `Simulation`, and `Constraints` to contain your source code.
3. Implement your sequencer and pixel generator designs in SystemVerilog.
4. Create a top-level module that contains the sequencer and pixel generator modules that implements the input and output ports shown in Figure 1.
5. Create a testbench to simulate your top-level module to verify that it correctly generates the necessary timing sequence and pixel values. Your testbench need not be self-checking, but you will find that the simulator is invaluable for finding mistakes before you attempt to debug the hardware. **Demonstrate your simulation to the instructor before attempting to implement your design on the FPGA**.
6. If not already connected, connect the LED Matrix unit to the Nexys4 FPGA board using the information in Figure 6 of the LED Matrix Overview.
7. Debug you circuit and verify that the display on the LED Matrix meets the requirements for the design. **Demonstrate your working design to the instructor**.

8. Create screen captures of your simulation on two different time scales to show (1) the timing for a single row; and (2) the timing for display of all 8 rows. **Include these screen captures in your report**.
9. Include in your report a finalize version or your **block diagram and SystemVerilog listings for all modules used in your design**, including modules that you used in previous labs or downloaded from the Moodle page. Format all SystemVerilog listings using a fixed-width font and single-line spacing. You need not include a listing of your constraints file.
10. Commit your corrected SystemVerilog code to your Git repository and push your repository back to the GitLab server.

5. Report

You must submit an electronic copy of your lab report in PDF format on Moodle. The name of your PDF file should be of the form

"`Lab02_Report_LastName1_LastName2.pdf`"

Be sure to label each section and organize them in the order given. Messy or disorganized labs will lose points.

Each lab report should have a "scribe" (i.e., the person who writes most of the report). The scribe should alternate every lab. Please clearly indicate in the title block of your report the names of the lab group and the name of the scribe.

Your report should include the following sections:

1. **Introduction** – write a brief paragraph describing what you did in the lab. This should be a summary written in your own words; it is not acceptable to copy text for this from the lab handout.
2. **Design** – Describe the details of how your design's organization and function, including block diagrams, state transition diagrams (if any), module descriptions, and properly formatted code listings. You may re-use properly revised parts of your design plan for this section.
3. **Results** – Include any items specified in the "Procedure" section. These items may include screen captures of the SystemVerilog simulator, a completed test plan, a summary synthesis report.
4. **Conclusion** – Briefly describe any difficulties that you encountered and how you resolved them. Also, if you have any suggestions for how to improve this lab, please include them here.
5. **Time Spent on Lab** – Please indicate how many hours you spent on this lab (including time during the lab period). This will not affect your grade, but will be helpful for calibrating the workload for future offerings of ECE 212.

Each section should begin with a section heading that includes the section number and title in a boldface font as in the report for Lab 1. Be sure to label each section and organize them in the order given. Messy or disorganized labs will lose points.