LED Array Tutorial

This guide explains how to set up and operate the LED arrays that can be used for your final EE 271 project. This tutorial is directed towards the FYM12882AEG 8x8 LED array, but these concepts can be used to operate other LED arrays.

Internal Structure of LED Array

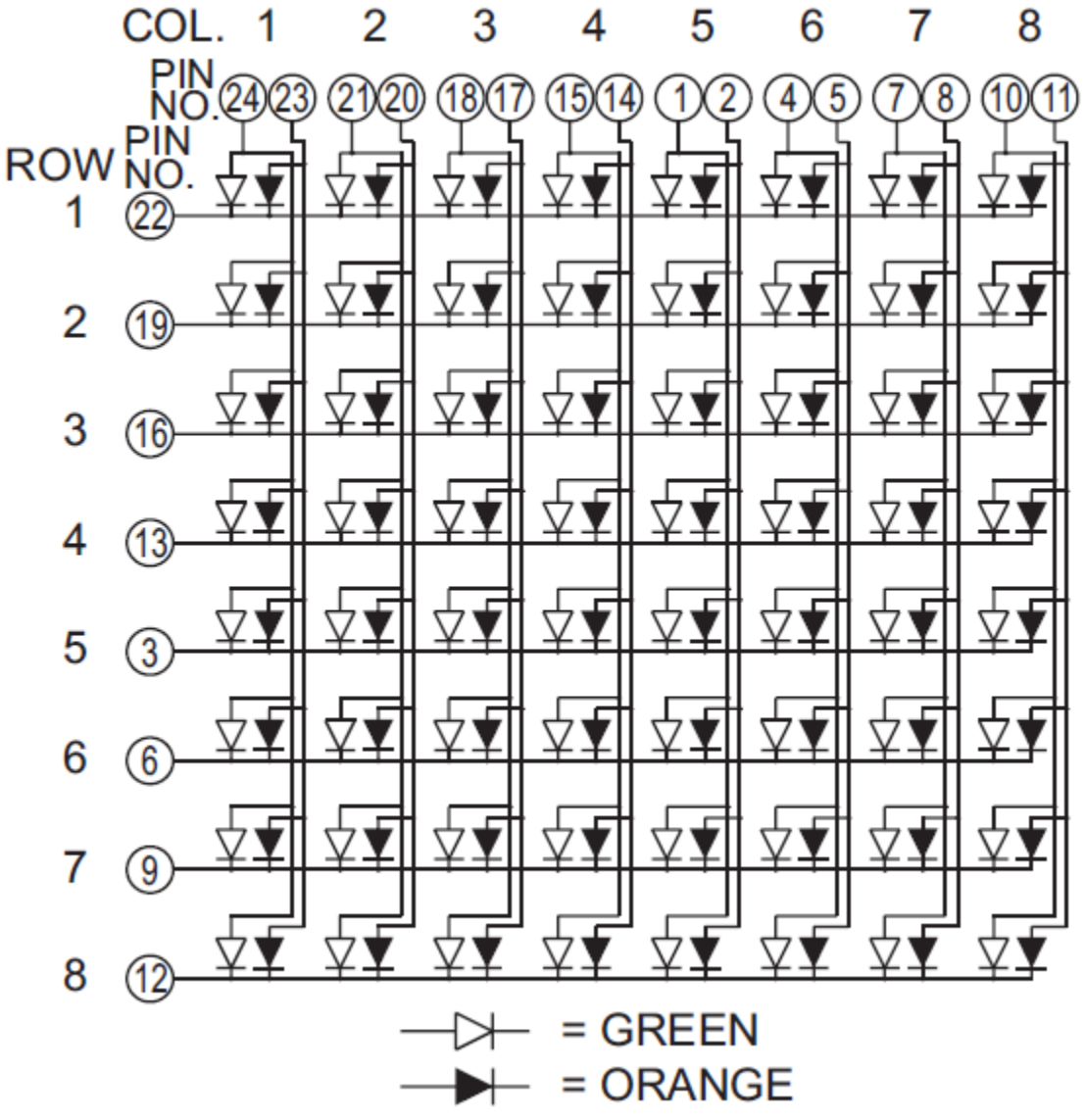

Figure 1: FYM12882AEG LED Array Circuit Diagram

Figure 1 displays the circuit inside the 8x8 LED array. As shown, there are eight rows and eight columns in the array. Every row-column pair contains one LED housing. But each row-column pair (there are 64 total) actually contains two LEDs, a green one and a red one (not orange as stated in the figure). To turn on an LED, a high signal (DE1 board output is 3.3V) must be connected to the anode while a low signal (ground) must be connected to the

cathode. As displayed in Figure 1, the pins connected to the LED anodes correspond to the array columns while those connected to the cathodes correspond to the array rows. Thus, we will refer to the rows as “sinks” since they are sinking or receiving current, while we will refer to the columns as “drivers” since they are setting the voltage which drives the LEDs to turn on. Each column actually contains two drivers, one for green and one for red. Figure 1 shows the mapping of pin numbers to rows and columns. The actual locations of the pins is discussed next.

Pin, Row, and Column Locations

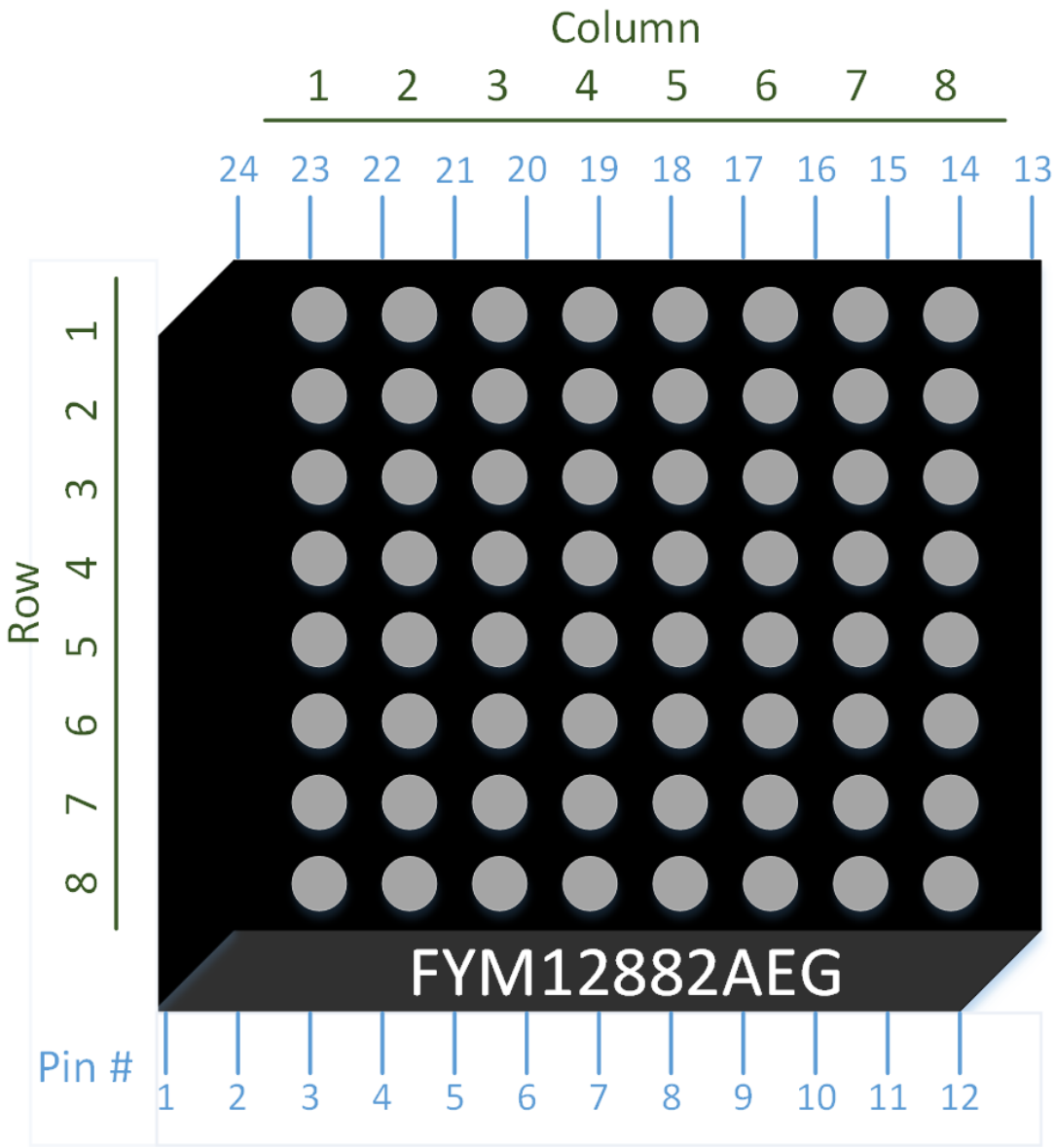

Figure 2: FYM12882AEG LED Array Pin, Row, and Column Locations (Top View)

Figure 2 depicts the 8x8 LED array from an overhead view. The part number (in this case FYM12882AEG) is only written on one side of the array, as shown. If your LED array is oriented in the depicted position, the pin, row, and column numbers are laid out as they are in Figure 2.

Breadboard Wiring

The FYM12882AEG has feet on each corner which prevent it from being inserted into a breadboard. The plastic is rather soft, so these feet may be carefully removed using a sharp knife. Additionally, you may need to bend the pins inward slightly to insert the array into the breadboard. It can be useful to run wires on the breadboard underneath the array to access the pins, since the 8x8 arrays are often the exact width of the breadboard's inner region. Without doing so, it can be difficult to access the nodes that the LED array pins are connected to. The wires may be directly connected from the LED array to the DE1_SoC's GPIO pins as the DE1_SoC has built in resistors on each GPIO pin for protection. Ordinarily, connecting an LED directly to a digital output would damage the LED and/or the device connected to it, but these built in resistors are of an appropriate value to limit the current through the LED.

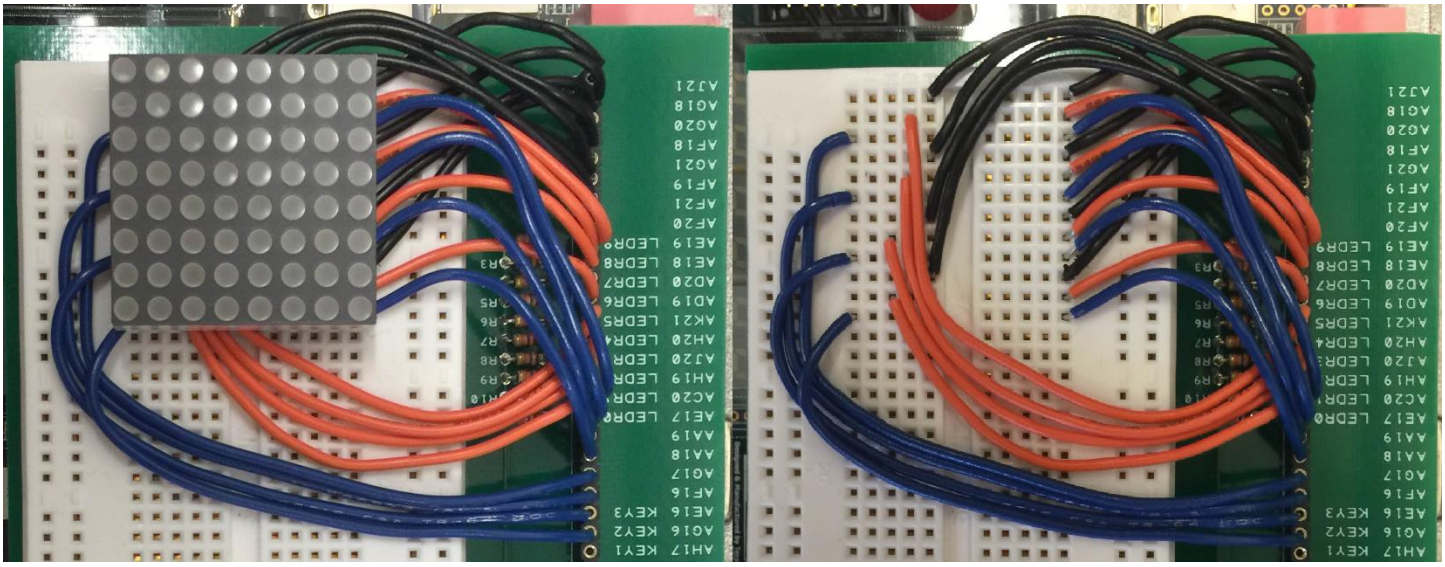

Figure 3: Example of Wiring the Breadboard

Figure 3 shows a possible wiring layout. The black wires are red LED column drivers, orange wires green LED column drivers, and blue wires row sinks. The LED matrix is oriented with its part number on the left.

Implementation Method

An arbitrary image may be displayed on the LED array by taking advantage of persistence of vision. The idea is to activate only one row at a time, showing the appropriate pattern for each row using the red and green column drivers. By cycling quickly through the rows, the effect is indistinguishable from having each LED controlled independently, effectively allowing us to use all 128 LEDs with only 24 wires. The human eye is unable to detect any flickering if the frequency is sufficiently high. Figure 4 below demonstrates how this may be

achieved in hardware, and a SystemVerilog code sample to implement this design is given later.

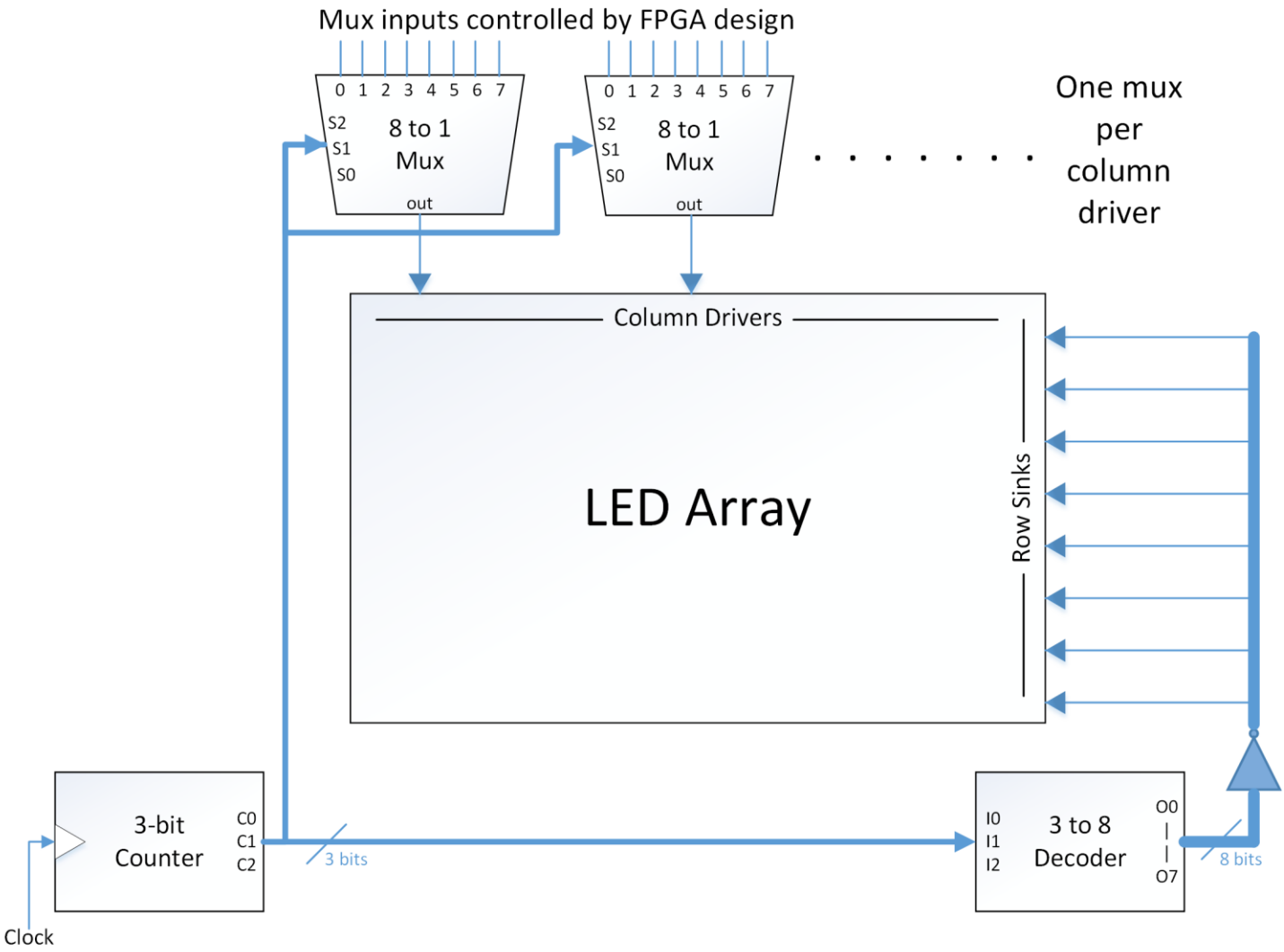

Figure 4: Suggested Implementation of 8x8 LED Array

To implement a matrix driver in hardware, we use a 3-bit counter to control a decoder and several multiplexers. The 3-bit counter continuously counts from 0 to 7, incrementing once each clock cycle, and outputs the current count in binary representation as three bits. When the count is at 7 and increments up one, it cycles back to 0.

The 3-bit output from the counter is the input to the 3 to 8 decoder. This decoder has eight outputs which at any given time are all set to low except one of them. The output that is set to high depends on the 3-bit input. For example, if the decoder's input interpreted as binary is 1, output O1 will be high while the other seven outputs will be low. And if the decoder's input is 6, output O6 will be high. However, we actually want one output set low and the rest high to enable one row at a time, which is the opposite of this decoder's output.

This is because connecting a low signal to a row sink enables it. So to get our desired result we invert the decoder outputs and connect them to the row sinks as shown in Figure 4.

An 8 to 1 multiplexer (mux) is connected to each column driver that will be used. The 3-bit output from the counter is connected as the select bits of each mux. This causes the mux to select from a different input (0 – 7) for each count value. Since the same count values are also controlling the decoder, the mux selection is synchronized with the row enabling. For example, when the count is 7, row 8 will be enabled and each mux will select input 7. Each of the eight mux inputs contains the value (high or low) to be driven to the connected column while its respective row is enabled, to turn the LED on or off. The inputs to the muxes should be connected to the rest of your design so that their values can be changed according to your project.

The counter, decoder, and multiplexers can be implemented with your FPGA using Quartus II. The decoder and multiplexer outputs can be assigned as GPIO outputs (GPIO_0[35] to GPIO_0[0] available) to access them externally from the DE1 board and connect them to the LED array.

SystemVerilog Implementation

An example of an LED matrix driver in SystemVerilog is given below. This example makes use of 2D arrays which comes with several advantages. The first is that a 2D array is, conceptually, a good structure for representing a 2D pattern of lights. Also, a 2D array simplifies the task of interpreting simulation results as the array will appear in Modelsim in its natural order which reflects what one would see on the LED matrix itself (provided the wiring is correct).

```
module led_matrix_driver (clock, red_array, green_array, red_driver,
            green_driver, row_sink);
      input clock;
      input [7:0][7:0] red_array, green_array;
      output reg [7:0] red_driver, green_driver, row_sink;

      reg [2:0] count;

      always @(posedge clock)
            count <= count + 3'b001;

      always @(*)
            case (count)
                  3'b000: row_sink = 8'b11111110;
                  3'b001: row_sink = 8'b11111101;
                  3'b010: row_sink = 8'b11111011;
                  3'b011: row_sink = 8'b11110111;
                  3'b100: row_sink = 8'b11101111;
                  3'b101: row_sink = 8'b11011111;
                  3'b110: row_sink = 8'b10111111;
                  3'b111: row_sink = 8'b01111111;
            endcase
```

```
      assign red_driver = red_array[count];
      assign green_driver = green_array[count];
endmodule
```

This SystemVerilog code contains the same pieces as described in the hardware diagram in Figure 4. A counter, `reg [2:0] count`, is incremented every clock cycle in the `always @(posedge clock)` block. An inverting 3 to 8 decoder sets the output of the row sinks in the `always @(*)` block. And, instead of individually instantiating sixteen separate multiplexer modules, the counter is used to index into the first dimension of each input array, effectively selecting out the desired row and assigning that to the column drivers. This is done in the two `assign` statements at the end.

Frequency Considerations

Cycling the LED array rows at 50 MHz (the speed of CLOCK_50), causes the LEDs to appear very dim. This could be because the capacitors at the GPIO pins do not have enough time to fully charge or discharge when their values are being cycled so fast. Lowering the frequency will make the LEDs brighter. Just make sure that your cycling is still fast enough that the LEDs appear to be continuously lit.

The LEDs should be bright enough for easy viewing when cycled at or below 1 MHz. Cycling slower than 1 MHz will make them slightly brighter.

You can use the clock divider module to get a slower clock which will slow down the cycling, but remember you should only use one clock for your whole design. So if you're using a clock divider make sure this clock is fast enough to be used for the rest of your project.

If you want to use a faster clock (like CLOCK_50) for your project, just extend the length of the 3-bit counter to more bits, and use the upper bits of that counter. That is, if you use the upper 3 bits of a 4-bit counter, it will change only every other clock cycle. If you use the upper 3 bits of a 5-bit counter, it will change only every 4^{th} clock cycle, and so on.