CMPE 121L: Microprocessor System Design Lab

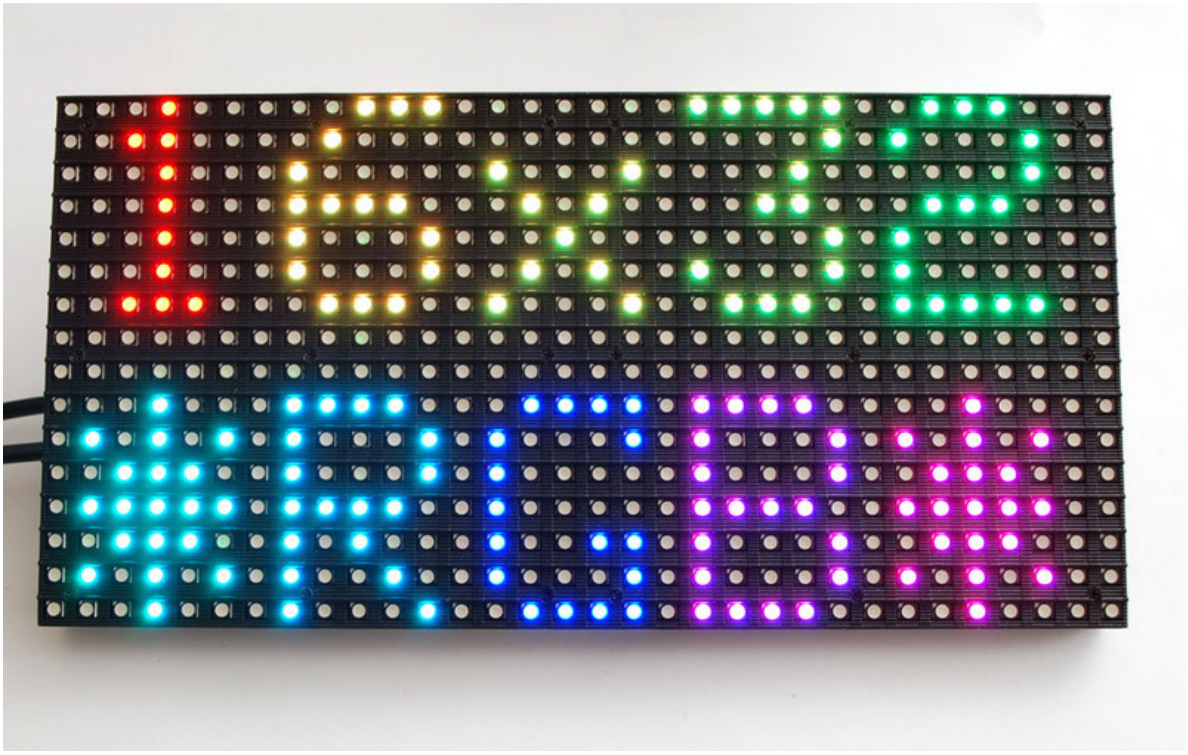
Fall 2016

Lab Exercise 5

Check-off Due in lab section, week of November 7, 2016

Report due: Within 3 days after checkoff due

In this project, you will learn to design a controller for the 16x32 RGB LED panel display included in your parts kit using row-column multiplexing. You will explore two different ways of designing the controller: (i) interrupt-driven refresh, and (ii) hardware controller.



This display has a total of 1536 LEDs (16 rows x 32 columns x 3 colors). These are arranged in a row-column format. For this exercise, you need to connect only the LEDs in a 4x4 grid formed by rows 1-4 and columns 1-4. This requires controlling a total of 48 LEDs (4 rows x 4 columns x 3 colors).

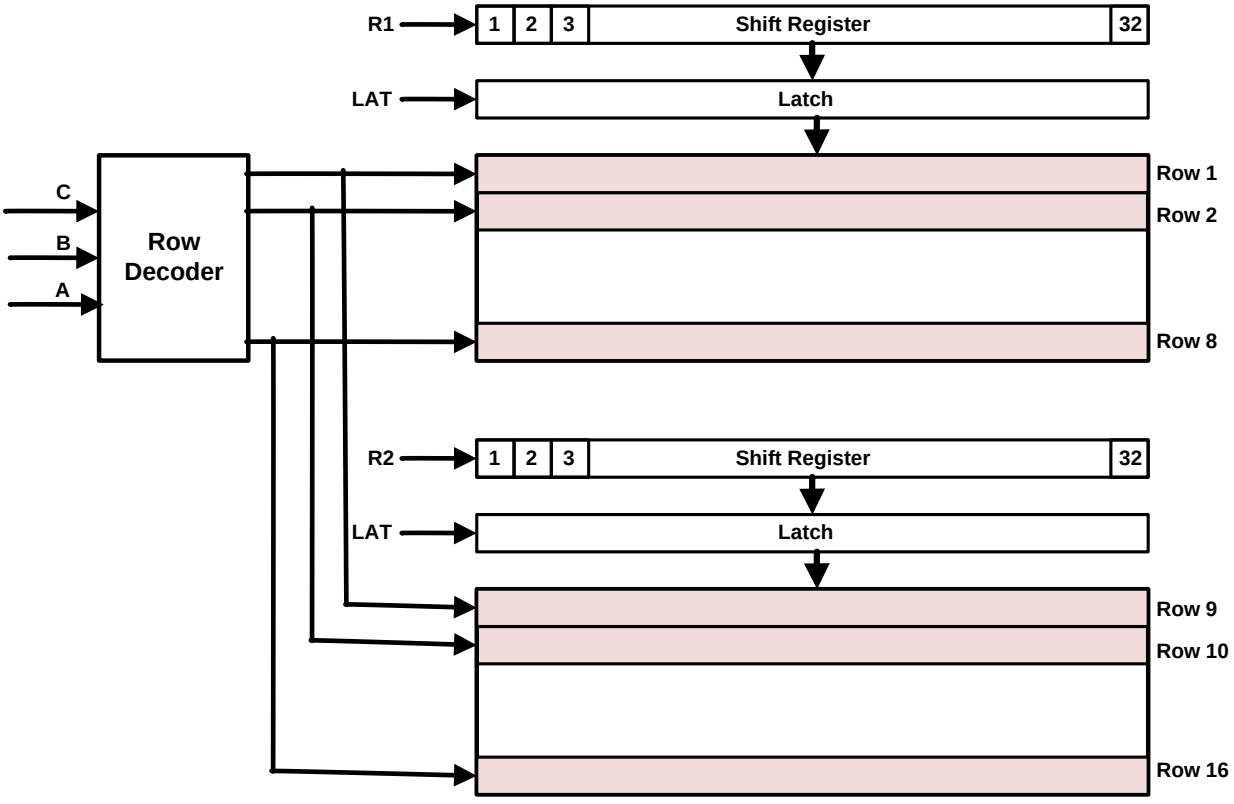
There is no datasheet available for this panel, but there is a good description of how it works on the Adafruit Web site (https://learn.adafruit.com/32x16-32x32-rgb-led-matrix/how-the-matrix-works). I have also provided below a step-by-step approach to interfacing it to the PSoC-5 and testing it.

Controlling the LED matrix

The LEDs of each color plane are organized as a two-dimensional matrix with 16 rows and 32 columns. A block diagram of one of the planes (the red LED plane) is shown on the next page.

The 16 rows are divided into two sections: The upper section consists of rows 1-8 and the lower section rows 9-16. The panel can only display one row in each section at a time. The row to be displayed is selected by the three row select inputs A, B and C (when CBA = 000, rows 1 and 9 are displayed; when CBA = 001, rows 2 and 10 are displayed; and so on). To display a pattern on the panel, the controller must cycle through the rows, displaying one pair of rows at a time. If this is done fast enough, the pattern will appear stable to the eye.

For each row, the pattern to be displayed on the 32 LEDs comprising its columns is sent to the display serially through the inputs R1, G1, B1, R2, G2 and B2. R1 is the serial input to be used for shifting in the data for the red LEDs in rows 1-8, and R2 is the serial input to be used for shifting in the data for the red LEDs in rows 9-16. G1/G2 and B1/B2 are used in the same manner to control the green and blue LEDs, respectively. Each serial input is connected to a 32-bit shift register, whose parallel outputs are used to control the LEDs along the columns.



To display a pattern on the 32 red LEDs associated with a row, the pattern must be shifted in one bit at a time using the R1/R2 inputs and the CLK input (each pulse on the clock input shifts in one bit). To avoid the display pattern from changing while the bits are being shifted in, the parallel output of the shift register is passed through a latch before driving the LEDs. This latch can hold a previous pattern while a new pattern is being shifted in. When the entire pattern has been shifted in, the latch can be enabled to replace the entire pattern for the 32 LEDs all at once.

Thus, displaying a 32-bit pattern on each row of red LEDs consists of the following steps:

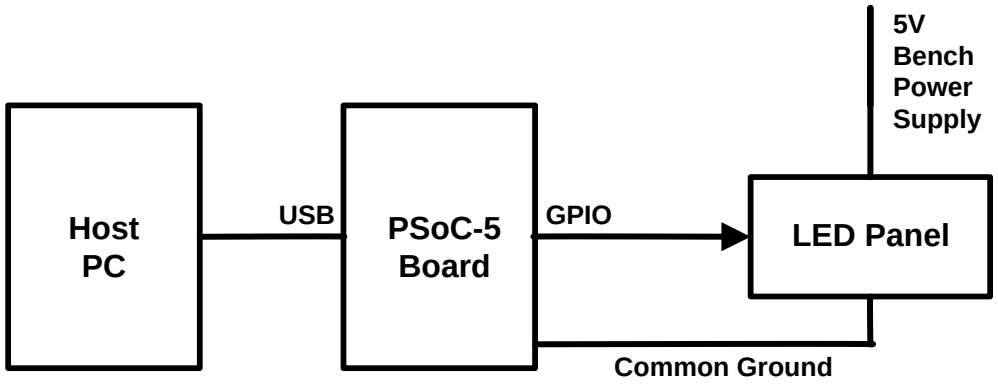
1. Select the desired row by setting the corresponding pattern on the row select inputs A, B, C.
2. Keep the output latch of the shift register closed by setting the LAT input low.
3. Shift in the pattern for the row one bit at a time using the R1 input (for rows 1-8) or R2 input (for rows 9-16), and the CLK. Each pulse on the CLK input shifts in one bit.
4. After the entire pattern has been shifted in, set the LAT input high and back to low to transfer the pattern to the latch.
5. The OE input must be set low to enable the display. Setting it high blanks the row being displayed.

The same sequence applied to the green and blue LEDs. Note that data can be shifted in through all the serial inputs (R1, R2, G1, G2, B1, B2) in parallel, so 32 clock cycles are adequate to load an entire row of all 3 colors in each section of the panel (a total of 32 x 2 x 3 = 192 LEDs).

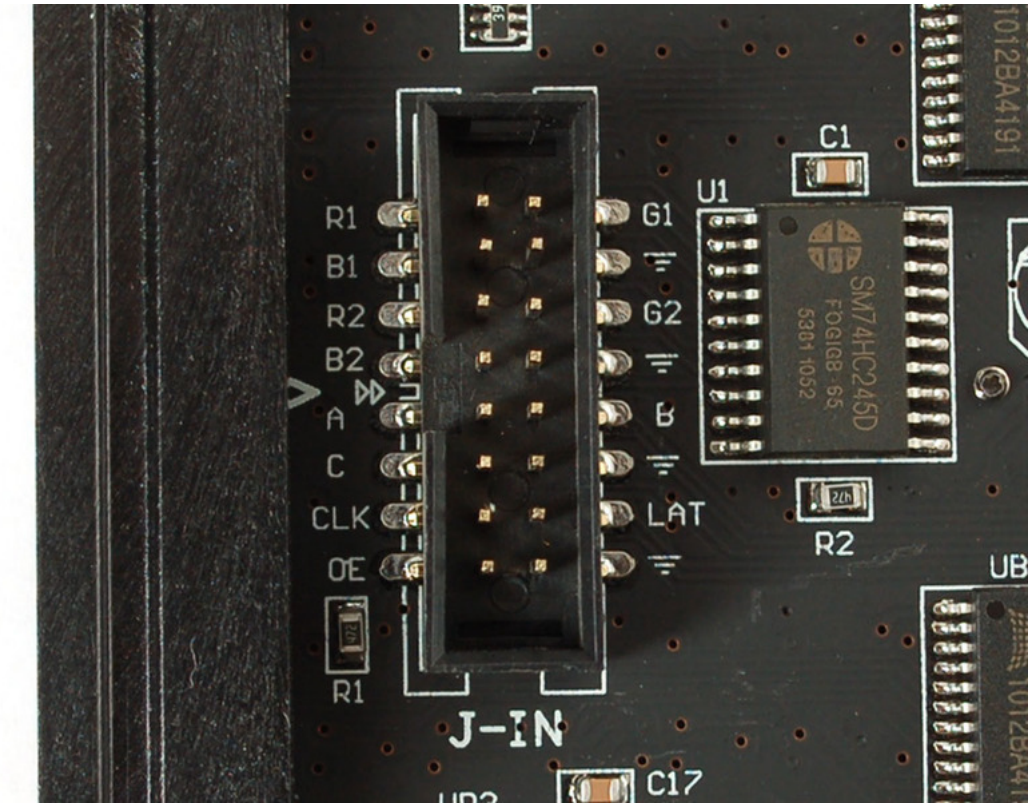
Connecting the LED Panel to the PSoC-5

The LED panel needs a 5V power supply. It consumes more than 2A when all the LEDs are on, which is well over the maximum current that can be supplied over USB (Even when using a part of the LED matrix, it is not safe to connect it to USB power). You should use the bench power supplies in the lab to power them. The following diagram illustrates the connections. In the diagram, the PSoC-5 is powered from USB, while the LED panel is powered from a separate 5V power supply. Please take the following precautions to avoid damaging the PSoC-5 and the LED display:

1. Set the power rail jumpers (J10 and J11) on the PSoC-5 board to the 5V setting.
2. Turn on your PSoC-5 first. Do not turn on the power to the LED panel until you are ready to run the program.
3. Double-check the polarity of the power leads before turning the power on.
4. If the bench power supply has the capability to limit the output current, use it to protect the system from an accidental overload.
5. Turn off the power to the LED matrix before turning off power to the PSoC-5 board.



Testing the LED Panel



To perform a basic test of the LED panel, proceed as follows:

1. Use the ribbon cable and jumper wires to connect PSoC-5 GPIO pins to the panel. The ribbon cable needs to be attached to the connector on the panel marked “IN”. You need to connect the following inputs of the panel to the PSoC-5:
    a. Serial inputs R1, G1, B1
    b. Row select inputs A, B, C
    c. Clock input CLK
    d. Latch enable input LAT
    e. Output Enable OE (active low)

    You do not need to connect the R2, G2 and B2 inputs, as we will not be using rows 9-16 of the panel.
    Allocate GPIO pins for the connections to the LED panel and configure them as outputs. Define Control Registers in the design to drive these outputs.

2. Using a test program, set R1 to high and all other GPIO pins to low.
3. Generate 32 pulses on the CLK input to the panel by setting it high and low alternately.
4. Generate a pulse on the LAT input by setting it high and back to low. You should now see all the red LEDs in Row 1 light up.
5. Repeat the steps by changing the setting of A, B, C to select a different row.
6. Repeat the steps for all 3 colors, using the G1 and B1 input instead of R1.
7. Now try to display the pattern red, green, blue, red, green, blue, … on some row.

Now you are ready to build a controller for the LED panel. To display a pattern on the entire panel, you will need to scan the LEDs one row at a time, and refresh them at a frequency high enough to eliminate flicker. This can be done in three different ways.

1. Using a timer to interrupt the microcontroller at periodic intervals, and refreshing the next row within the ISR.
2. Designing a custom hardware controller out of the UDB blocks to refresh the LEDs.
3. Using the DMA controller in conjunction with a timer to periodically transfer the refresh pattern from memory to the LEDs (we will not attempt to do this).

In this exercise, you will implement the first two methods and compare them.

Part 1: Interrupt-Driven Refresh

Define three two-dimensional arrays in your main program:

uint8 red[4][4], green[4][4], blue[4][4];
(Note: the display rows and columns are numbered 1 – 4, while the array indices range from 0 through 3).

Your objective is to turn on the red LED at row *i* and column *j* when the element red [*i*-1][*j*-1] is non-zero and turn it off when red [*i*-1][*j*-1] is 0, and similarly for the other two arrays. This allows any combination of the 48 LEDs to be on. Note that, although many LEDs may appear to be on at the same time, only one row of each color will be actually on at a time. The rapid refresh will make multiple rows to appear simultaneously on.

Drive the GPIO pins connected to the LED panel from control registers in your design. Configure a timer to generate an interrupt every millisecond. In the ISR, read from the arrays defining the LED states and write into the control registers to perform the refresh.

Part 2: Hardware Controller

In this part, you will design a hardware controller out of UDB blocks to refresh the display. Instead of storing the LED state in memory arrays, you need to now store it in control registers (you will need 4 rows x 4 columns x 3 colors = 48 flip-flops. These could be organized as sixteen 3-bit control registers.). The hardware controller should take its inputs from the 16 control registers and activate the row/column GPIO pins. The controller can be driven from a 1

kHz clock. The controller is essentially a state machine that cycles through the rows and refreshes their states. The state machine, in turn, can be designed from the LUT component of the PSoC Creator library.

There are three possible approaches to designing the hardware controller:

(a) Design the controller using logic primitives available in the PSoC Creator component library (gates, flops, LUTs, counters, multiplexers, etc.), and use the schematic tools to enter the design.
(b) Write Verilog code to describe your design. PSoC Creator will synthesize the Verilog design and will set up the UDBs to implement your logic.
(c) Use the UDB Editor tool in PSoC Creator to design the controller. The UDB Editor Guide under the help tab has examples on how to do this. This approach is not as flexible as (a) or (b), but can be simpler. It can also be used to develop custom components. For example, you can make the entire display controller a component that can be dropped into designs, just like a standard library component.

Option (a) is the recommended approach, as it only requires basic knowledge of logic design. Whichever approach you choose, start with the simplest controller (one color, one row, etc.) and enhance it step by step to its full functionality.

Part 3: PWM Control of LED Brightness (Optional, Extra Credit of 25%)

Even though each LED remains ON less than $1/8^{th}$ of the time, you will notice that the display is too bright for close viewing. You can control its brightness through PWM. (You can also use this method to control the relative brightness of the RGB colors, thus allowing many shades of colors to be displayed, but this takes a lot more logic and will probably require an FPGA).

Enhance your design from Part 2 to allow 256 levels of brightness. Level 0 should correspond to all LEDs being fully dark and level 255 should correspond to the brightness level obtained in Part 2. All LEDs that are ON will have the same brightness. Then use a potentiometer and ADC to set the 8-bit value for the brightness level. Note that you will need a much higher frequency for your refresh clock to avoid flicker at low brightness levels.

CHECKOFF:

For each part, show that are able to set the contents of red, blue and green LEDs to light up any pattern of LEDs in the 4x4 grid on the panel. The LEDs should not show any flicker or any significant bleeding of colors from an *on* LED to an adjacent *off* LED (called *ghosting*).

What to submit?

- External schematics showing how the display module is interfaced to the GPIO pins (please provide a clean hand-drawn diagram or a schematic from Eagle). The schematics should show pin numbers of all connections on each side and should be detailed enough for someone to wire up the circuit.

- Internal schematics from PSoC Creator for Parts 1 and 2.
- Descriptions of your designs
- Discuss the advantages and disadvantages of the two methods. Which one will you use for your final project if you have to extend the design to control the entire 16x32 RGB array?
- One of the limitations of these designs is that all the LEDs have the same duty cycle (1/8). This limits the number of shades available to 8, and you will not be able to get some of the colors in the image on Page 1. How will you enhance your design in Part 2 to support 4096 different colors?