16 x 16 Bicolor LED Expansion Board

Board & Driver Documentation

Introduction & How it Works

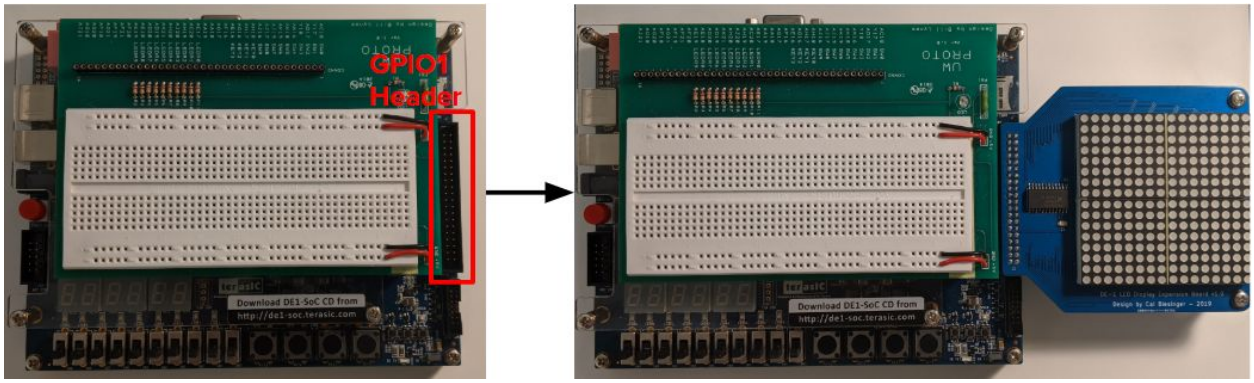
The 16 x 16 Bicolor LED Expansion Board (LED Board) is a 16 x 16 display with each pixel containing both a red and a green LED. Every LED is individually controllable, for a total of 16 x 16 x 2 = **512** individually controllable LEDs. You can set any pixel to be either off, red, green, or turn both red and green on for a yellow-orange-like color.

The LED Board is plugged into the GPIO1 header on the DE1-SoC board (see **Placement** section). Notice the GPIO1 header contains just 36 pins. We can use these to control all 512 LEDs using a persistence of vision illusion. At any instant of time, we actually illuminate only one row of the LED Board, but if we cycle through all the rows fast enough, our eyes perceive a display that looks like a steady image. The speed at which we cycle through the rows of the LED Board is called the **scan speed**.

This scanning is accomplished using a given driver module, **`LEDDriver`**, which internally switches on only one row of the display at a time, displaying the correct pattern, then moves onto the next, repeating this indefinitely. Using this driver module abstracts this behavior so you, the programmer, can interface with it as if it were just a normal array of outputs. These outputs are two 16x16 arrays named `RedPixels` and `GrnPixels` in the driver and demo code.

Placement

Carefully connect the LED Board to the DE1-SoC board as shown in the pictures below. The connector has a key/notch in the center so there is only one correct orientation. Be sure the connector is fully seated in the socket.



Driver & Demo Code

Download the associated led_board.zip and open the DE1_SoC project (.QPF) file. You will see a couple of SystemVerilog files:

- **`DE1_SoC.sv`** - top level module to program to the DE1-SoC board
- **`LEDDriver.sv`** - contains the driver module `LEDDriver`
- **`led_test.sv`** - a simple test module to display a static pattern
- **`clock_divider.sv`** - standard clock divider module to generate necessary system clock

Set DE1_SoC.sv as the top level entity, compile, and program the DE1-SoC board. You should see a static pattern illuminate the entire board. Hold down `KEY0` to turn all LEDs off.

Carefully read **`DE1_SoC.sv`** and **`led_test.sv`** to understand how to set up the driver and manipulate the LED Board yourself. Feel free to also look around in **`LEDDriver.sv`**, but you do not need to make changes to it.

To get started on your own design, copy these SystemVerilog files into your own project and replace the instantiation of `led_test` in `DE1_SoC.sv` with your own code. Remember that once the `LEDDriver` module is instantiated in your top-level entity, you don't need to 'touch it' again. Simply control the values of `RedPixels` and `GrnPixels` to manipulate the display, similarly to how you used `LEDR` in the past.

`RedPixels` and `GrnPixels` are declared as 16x16 unpacked arrays, corresponding to the 16x16 pixel grid of the LED Board. Set any bit in these arrays to 1 to turn on the corresponding pixel and color, or 0 to turn it off. You may find reviewing multidimensional array indexing, concatenation, and bit manipulation helpful in controlling these arrays in your own project.

Troubleshooting

In your top-level entity, verify that `RedPixels` and `GrnPixels` are behaving correctly in ModelSim. If the display is not working, start by resetting the DE1 board, and check to make sure the display shows an orange line on the bottom row of the display. If not, check that it is connected correctly and fully seated in the socket. Try re-running the demo code from scratch to verify it is not a hardware issue.

Next, check to make sure EnableCount is set to true, and that the driver is being reset before attempting to be used, but that the reset line is low during operation.

Verify that the display is connected to GPIO1, not GPIO0 as you may have used in the past.

If the display is noticeably flickering, try a faster system clock. If the display is too dim or it appears that pixel colors are blended, try a slower system clock.

Additional Documentation on LED Driver Parameters

The `FREQDIV` parameter is configurable if desired, but otherwise defaults to 0 and must be a positive integer. This allows you to change the overall scanning speed of the display without changing the input clock. If you find that the display appears to flicker, you may want to speed up the input clock or reduce this number. However, making the input clock too fast or `FREQDIV` too small will reduce the brightness of the display and cause red and green pixels to appear blended.

(This sets the number of clock pulses between the driver switching rows on the display, with the delay being 2 raised to the power of FREQDIV. The default of 0 used with a 3051 Hz clock will set this to 16384 clock cycles per line, which translates to 1/3051s. This means the entire display is scanning at about 190Hz.)

`GPIO_1` should be connected to the GPIO_1 header in output mode. To set this up, simply add "`output logic [35:0] GPIO_1;`" to your DE1_SoC (or top-level) module, and add "`GPIO_1`" to the list of parameters, similar to any of the other I/O connections.

`CLK` should be connected to whatever clock source is used for the rest of your design (since there should only be one). For best results, use clocks between 1.5 kHz and 100 kHz. If you are using the standard clock_divider module, these would represent output clock indices clk[14] to clk[9], respectively. The demo code uses a ~3 kHz clock, clk[13].

`RST` is the reset line for the driver. This should be connected to the same reset line as the rest of your system, and should be set high then low at least once before trying to use the display, as it may not behave correctly at startup without a reset signal.

`EnableCount` should be high by default. This can be used if you'd like to temporarily pause scanning of the display. When this is set to low, the display will stay on the current line, and continue displaying the pixels on that line. When set to high, the display will continue scanning.

`RedPixels` and `GrnPixels` should both be a set of 16x16 connections, with each one corresponding to its respective pixel on the screen. To determine what is shown on the display, simply set these lines. Both red and green can be on for the same pixel to display a yellow-orange colour.