Adafruit's 16x32 LED Matrix Guide for BeagleBone

Written by: Janet Mardjuki

1. Introduction

This guide will provide guidance to use the 16x32 LED Matrix that is sold by Adafruit. LED Matrix can produce different RGB colour on 'screen' itself, but it is not 'true RGB' as the R, G, and B input only has one bit (only 0 or 1) associated with each signal, therefore, the selection is pretty limited. The LED Matrix has no built-in PWM; therefore, the frame has to be quickly 'refreshed' for every period of time.

You might want to read more about how the LED Matrix works by reading the following link:
https://learn.adafruit.com/32x16-32x32-rgb-led-matrix/how-the-matrix-works

2. Setup

2.1 Wiring-up the wires

You can take a look at the following link if you want to understand in detail what each input represents:
https://learn.adafruit.com/32x16-32x32-rgb-led-matrix/connecting-with-jumper-wires
From previous link, you learned that the ribbon wire ends are flipped, which we can get the following positions for the other end of the wire to connect with the BeagleBone:

G1	R1 (Red wire)
GND	B1
G2	R1
GND	B2
B	A
D	C
LAT	CLK
GND	OE

Figure 1: Ribbon wire signal position to be connected with GPIO

The following table shows us what each signal represent and how are they used for the LED Matrix:

Label	Signal	Description
R1	Red1	Red colour signal for the top half of the matrix (x-index: 0-7)
B1	Blue1	Blue colour signal for the top half of the matrix (x-index: 0-7)
G1	Green1	Green colour signal for the top half of the matrix (x-index: 0-7)
R2	Red2	Red colour signal for the bottom half of the matrix (x-index: 8-15)
B2	Blue2	Blue colour signal for the bottom half of the matrix (x-index: 8-15)
G2	Green2	Green colour signal for the bottom half of the matrix (x-index: 8-15)
A	Row A	MSB of the row bits for the row selection
B	Row B	Middle bit of the row bits for the row selection
C	Row C	LSB of the row bits (for the 16x32 Matrix) for the row selection
D	Row D	LSB of the row bits (for anything bigger than 16x32 Matrix) for the row selection

CLK	Clock	Ends of each bit of the data (each pixel)
LAT	Latch	Ends of row of data
OE	Output Enable	LED on/off (enable/disable), for row transition
GND	Ground	Signal to ground

Figure 2: Ribbon wire signal position at the end

It is highly recommended to use different colour of wires for different signal. General rule of thumb, always use black wire for ground. The author of this guide, also prefers to connect all the jumper wires to the ribbon wire first and GPIO later than connecting the jumper wires to ribbon wire and GPIO one-by-one at once.

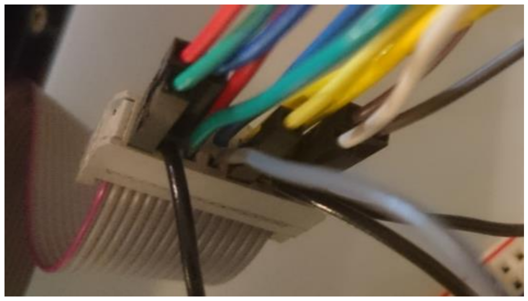
Figure 3: Ribbon wire attached with jumper wires

In general, any 12 GPIOs that are available on the BeagleBone are good for this LED Matrix. Please take a look at http://beagleboard.org/support/bone101 or http://target_ip/ (for example, http://192.168.2.2/ which will redirect to http://192.168.2.2//support/bone101) to figure out the mapped Linux GPIO number. The author prefers to keep all the GPIOs near by each other and at the corners; the following are the GPIOs the author used: (the pin numbers are in the brackets '[]')

R1 [8]	G1 [80]
B1 [78]	G2 [79]
R2 [76]	B [77]
B2 [74]	LAT [75]
A [72]	CLK [73]
C [70]	OE [71]

Figure 4: GPIO mapping to the LED output

Note: If you need to use the HDMI, you would not be able to use above mapping, as there will be conflict with the pin usage.

In addition to those signal pins mentioned above, you would also need to connect the three GND signal wires and a D signal wire to the GND pins on BeagleBone, which should be enough as it has six GND in total.

If you decided to use any kind of cape you will have to make sure that the GPIOs you will be using are not used by the cape, which can be done by reading the cape schematics, and check both P8 and P9 pins usage.
If the cape you wanted to use does not have a female/ tall male pin header attached on top, you can try tall pin header to connect the cape and BeagleBone (shown on figure 5), and

use the extra gap you just created to attach the wires; last resort is to solder the pins you need (not recommended unless you know how to do it or you can ask someone who are experienced).

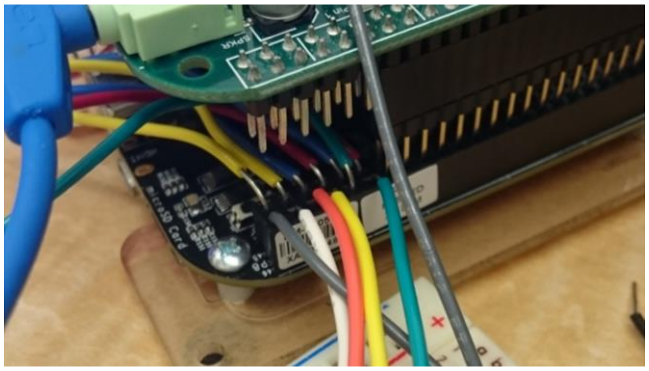
Figure 5: All the jumper wires are connected to respective the GPIO pins

After attaching the jumper wires, you can plug the other end (the unused one) of the ribbon wire to the LED Matrix Input (labeled with IN).

2.2 Powering on
You would need to supply the matrix with +5V power; you can just attach the Molex header to the power pin, and connect and plug in the power supply.

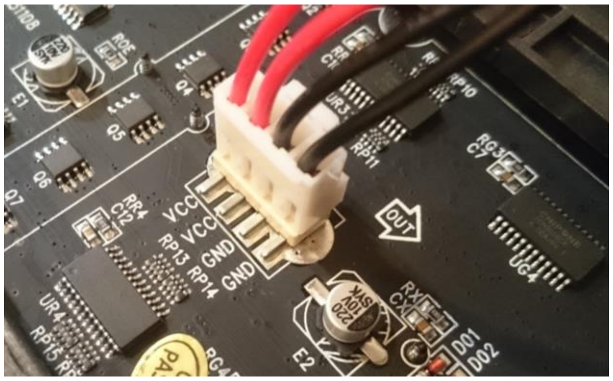

Figure 6: Connecting the Molex header

If you have different power model, please refer to the following link:
https://learn.adafruit.com/32x16-32x32-rgb-led-matrix/powering
After powering the LED Matrix, you might see the matrix lights up and displays different pattern.

3. Testing the LED Matrix
3.1 Disabling the HDMI
If you decided to use this guide's GPIO pins mapping, you would have to disable to the HDMI output overlay as those are the default setting, and it conflicts with the GPIO. You can skip this part if you decided to use other pins.

1. First, you will have to edit the uEnv.txt: (You can use any editor that you like)
```
# vim /boot/uboot/uEnv.txt
```

2. Find the line `##Disable HDMI` on the uEnv.txt, under that line it you will find the following line:

```
#optargs=capemgr.disable_partno=BB-BONELT_HDMI,BB-BONELT_HDMIN
```

3. Uncomment that line by removing the `#` sign in front of the line.
4. Save the file and exit the text editor.
5. Reboot the target

```
# reboot
```

6. After you gain your access back, check the slots loaded on BeagleBone

```
# cat /sys/devices/bone_capemgr.*/slots
0: 54:PF---
1: 55:PF---
2: 56:PF---
3: 57:PF---
4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
5: ff:P-O-- Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
6: ff:P-O-- Bone-Black-HDMIN,00A0,Texas Instrument,BB-BONELT-HDMIN
```

You should get the result above; both HDMI and HDMIN are unloaded in the BeagleBone which you can tell by checking the char `L` which are not in `ff:P-O--` for HDMI and HDMIN.

3.2 Writing the testing program

About 80% of this guide's test codes are converted from Python to C, and source is: https://learn.adafruit.com/connecting-a-16x32-rgb-led-matrix-panel-to-a-raspberry-pi/experimental-python-code

The test program in this guide need the general module created by Dr. Brian Fraser, or you can create your own `sleep_usec()` function, which basically let the program sleep for a number of ms.

The testing code `test_ledMatrix.c` and the `Makefile` will be provided in another link; you can read the comments to understand what the code does.

The following code snippet is the refresh method for that will be discussed later:

```
/**
 *  ledMatrix_refresh
 *  Fill the LED Matrix with the respective pixel colour
 */
static void ledMatrix_refresh(void)
{
        for ( int rowNum = 0; rowNum < 8; rowNum++ ) {
                lseek(fileDesc_oe, 0, SEEK_SET);
                write(fileDesc_oe, "1", 1);
                ledMatrix_setRow(rowNum);
                for ( int colNum = 0; colNum < 32;  colNum++) {
                   ledMatrix_setColourTop(screen[colNum][rowNum]);
                   ledMatrix_setColourBottom(screen[colNum][rowNum+8]);
                   ledMatrix_clock();
                }
                ledMatrix_latch();
                lseek(fileDesc_oe, 0, SEEK_SET);
                write(fileDesc_oe, "0", 1);
                sleep_usec(DELAY_IN_US); // Sleep for delay
        }
        return;
}
```

The function refresh above is a function to fill one frame of the LED Matrix at a time. Data to the matrix send at every rising edge, the program basically queuing up the value for the LED Matrix for the current row. For example, the program will be sending the bit for y-axis pos 0, then next clock, y-axis pos 1, then next clock, y-pos 2, and then keep going until y-axis pos 31. When you are done with the current row, you will have to turn off the output, as the hardware cannot light up more that one row, then latch telling the hardware that you are done with the current row. The LED can light up two lines at a time; one must be on the top eight and one on the bottom eight.

Note: The `open()` method getting each GPIO value in the `test_ledMatrix.c` are hardcoded with GPIO value; if you chose different GPIO you would have to change the value on `#define` and on the `open()` method.

After you compiled the program using `make` command and run the program on target, you should get the following displayed on the LED Matrix screen:

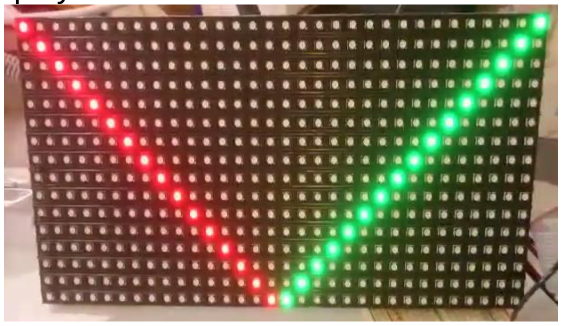
Figure 7: Final result in running the program

4. Troubleshooting

1. If the LED Matrix flickers a lot, try decreasing the delay, or you can try kernel code or use the PRU (would not be covered by this guide)
2. If the LED Matrix displays random pattern with random colour, probably you did not attach your GPIOs/wire properly
3. If only left half of the LED light up, you probably did not plug in the power properly
4. If only the top half/top bottom light up, check again your code whether you copied properly. If this does not work, you can try checking the wires for the RGB 1 or 2 depending on which portion does not get displayed properly.
5. If only some parts of the pattern get displayed, check back the 'row' signal wires, make sure they are attached to the appropriate GPIO
6. If none of the LED 'pixel' light up you can try the following method to fix it
    1. Try checking all the jumper wires connection, make sure all jumper wires are connected in the appropriate place
    2. Check back all the wires connected, make sure they connected to the right GPIO and GND
    3. If you are using the GPIO this guide use, check again for the HDMI overlay, make sure they are all disabled

4. If you have a DMM, you can check if your GPIO actually drive about 3.3V voltage when GPIO value is ‘1’. Clip the black wire to GND, and the Red to the GPIO you want to test (shown in the image below)

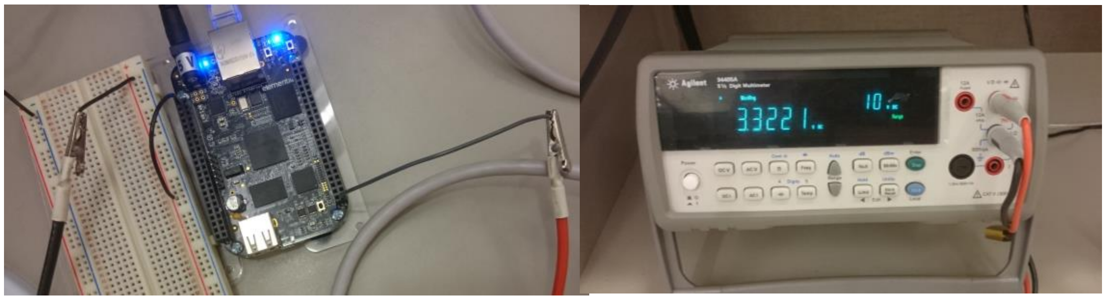

Figure 8: Checking the GPIO using DMM