Guide to connecting USB device to BBG

By: Vy Bui, Hyeonyoung Park, Jooyoung Lee

1. Install libusb and enable permissions

This guide can be used to connect any USB device, but the driver code is specifically for an XBox controller. Modify values and addresses accordingly.
In this step, we are going to install libusb on the host to compile the driver code for the controller on the target. libusb is a C library that our driver code can use to provide access to USB devices on the target. To cross-compile for the target, we need the armhf binary of libusb.

1. On the host, check if you already have the armhf architecture by running:
   (host) $ dpkg --print-foreign-architectures
2. This should print “armhf”. If you do not see it, add the architecture as follows:
   (host) $ sudo dpkg --add-architecture armhf
   sudo apt-get update
3. Run (host) $ dpkg --print-foreign-architectures to check if the architecture has been successfully added.
4. Now that you have armhf added, install libusb for target arm-linux-gnueabihf
   (host) $ sudo apt-get install libusb-1.0-0-dev:armhf
5. Find the vendor id and product id of your usb device that you wish to connect. Connect the device to your target. Then, you can use lsusb to list details about all the usb devices that are currently connected to your machine. Install lsusb, and then run the command:
   (bbg) $ sudo apt-get install usbutils
   (bbg) $ sudo apt-get update
   (bbg) $ lsusb
   This should print out details about the usb device that is connected to your bbg. Look for the vendor id and product id in this list of information.
6. On target, enable permissions for reading input from our USB device. Create a new udev rule:
   (bbg) $ sudo nano /etc/udev/rules.d/usb-permissions.rules
   (you can name the .rules file whatever you want.)
7. Add this line:
   (bbg) $ SUBSYSTEM=="usb", ATTRS{idVendor}=="045e", ATTRS{idProduct}=="028e", MODE="0666"
   0x045e and 0x028e are the *vendor id* and *product id* of an xbox controller. To add permissions for your own usb device, you must find your own vendor id and product id in step 5.
8. Reload udev rules for it to take effect: sudo udevadm control --reload-rules

You may need to unplug and replug your usb device for it to work.

Troubleshooting:

- If your linker cannot find libusb when you run make, make sure that your linker options include "-lusb-1.0". If that still doesn't work, specify the location: "-L/usr/lib/arm-linux-gnueabihf -lusb-1.0" in your makefile. The full command should be something like:
  $(CC_C) $(CFLAGS) $@.c $(DEPS) -o $(OUTDIR)/$@ -L/usr/lib/arm-linux-gnueabihf -lusb-1.0
- Make sure you have done steps 1-4 correctly.
- If you get the error "libusb_claim_interface() failed: Resource busy" when running the controller code on the target, there is likely an issue with the kernel driver. The module code should have already detached the device from the kernel driver, but if it does not work, you may need to do these steps (taken from this stackoverflow thread)
  run:
  (bbg) $ sudo nano /etc/modprobe.d/blacklist-libnfc.conf
  Add the following to the configuration file:
  blacklist pn533
  blacklist nfc
  save the file, then run:
  (bbg) $ modprobe -r pn533 nfc

2. Driver code

The following code snippet is the basic driver code for a usb device. Again, the VENDOR_ID and PRODUCT_ID constants are written for an XBox controller, as well as some of the specific driver code. However, you could reuse them for a different usb device as well.

```
#include <stdio.h>
#include <stdlib.h>
#include <libusb-1.0/libusb.h>

// change IDs for your USB device
#define VENDOR_ID 0x045e
#define PRODUCT_ID 0x028e
#define INTERFACE_NUMBER 0
#define ENDPOINT 0x81

int main(int argc, char **argv)
{
    libusb_context *ctx;
    libusb_device_handle *dev_handle;
    int error;
    ssize_t cnt;

    error = libusb_init(&ctx);
```

```
    if (error != LIBUSB_SUCCESS) {
        fprintf(stderr, "libusb_init() failed: %s\n", libusb_strerror(error));
        return 1;
    }

    // Use libusb to open device
    dev_handle = libusb_open_device_with_vid_pid(ctx, VENDOR_ID, PRODUCT_ID);
    printf("%d, %d",VENDOR_ID,PRODUCT_ID);
    if (dev_handle == NULL) {
        fprintf(stderr, "libusb_open_device_with_vid_pid() failed: device not
found\n");
        libusb_exit(ctx);
        return 1;
    }
    if (dev_handle != NULL)
    {
        libusb_detach_kernel_driver(dev_handle, INTERFACE_NUMBER);
        {
            error = libusb_claim_interface(dev_handle, 0);
            if (error != LIBUSB_SUCCESS) {
                fprintf(stderr, "libusb_claim_interface() failed: %s\n",
libusb_strerror(error));
                libusb_close(dev_handle);
                libusb_exit(ctx);
                return 1;
            }
        }
    }
    unsigned char data[20];
    while (1) {
        error = libusb_interrupt_transfer(dev_handle, ENDPOINT, data, sizeof(data),
&cnt, 0);
        if (error == LIBUSB_ERROR_INTERRUPTED) {
            break;
        }
        if (error != LIBUSB_SUCCESS) {
            fprintf(stderr, "libusb_interrupt_transfer() failed: %s\n",
libusb_strerror(error));
            break;
        }
        // Uncomment these lines to show the hexadecimal values registered by the
input of your usb device, which you can then use to read input.
        // printf("Received input: ");
        // for (int i = 0; i < cnt; i++) {
        //     printf("%x ", data[i]);
        // }
        // printf("\n");
```

```
        // these specific values are for an XBox controller's right thumbstick.
        if (data[8]==0xff && data [9]==0x7f){
            printf("up\n");
        }
        if (data[7] == 0x80){
            printf("left\n");
        }
        if (data[6]==0xff&&data[7]==0x7f){
            printf("right\n");
        }
        if (data[9]==0x80){
            printf("down\n");
        }
    }
    // clean up any interface usage.
    libusb_release_interface(dev_handle, 0);
    libusb_close(dev_handle);
    libusb_exit(ctx);

    return 0;
}
```

Guide to connecting 16x32 LED panel to BBG

This guide focuses on running code to light up an LED panel. More detailed guides can be found at the following links:
1.https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/links/files/2015-student-howtos/AdafruitLEDMatrixGuideForBBB.pdf
2.https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/links/files/2018-student-howtos/AdafruitLEDMatrixGuide.pdf

Our objective is to connect the LED panel's pins to the free BBG (Zen Cape) pins (P8 and P9) using female-to-female jumper wires. Note that these wires are not provided in the BeagleBone kit, so you may need to borrow or purchase them. For easier connection, tape the jumper wires in bundles.

Figure 1 demonstrates an example of pin mapping. These decisions are mostly for the developer's convenience, as the pins are adjacent to each other. We suggest using any of pins 45, 46, 44, and 43 for GND pins and pins 70-80 for the rest. It's easier to track the LED pin-to-GPIO mapping after plugging them in, so do this first and then create a mapping table like Figure 1. Update the GPIO #define variables in your LED display C module accordingly.

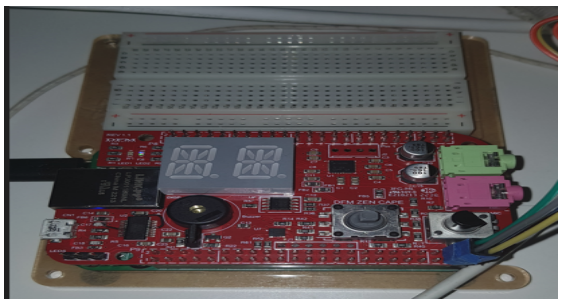
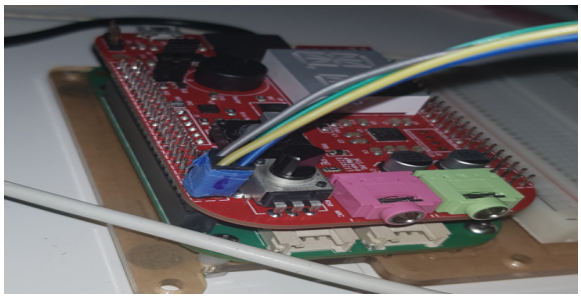
R1 [78]	G1 [70]
B1 [8]	GND [45]
R2 [71]	G2 [72]
B2 [73]	GND [46]
A [75]	B [74]
C [77]	D(GND) [43]
CLK [79]	LAT [76]
OE [80]	GND [44]

```
#define RED1_PIN 78     // UPPER
#define GREEN1_PIN 70
#define BLUE1_PIN 8
#define RED2_PIN 71 // LOWER
#define GREEN2_PIN 72
#define BLUE2_PIN 73
#define CLK_PIN 79
#define LATCH_PIN 76
#define OE_PIN 80
#define A_PIN 75
#define B_PIN 74
#define C_PIN 77
```

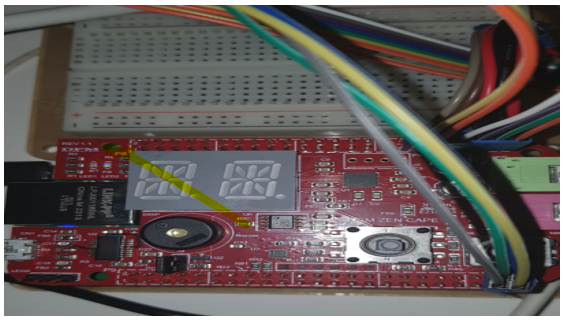
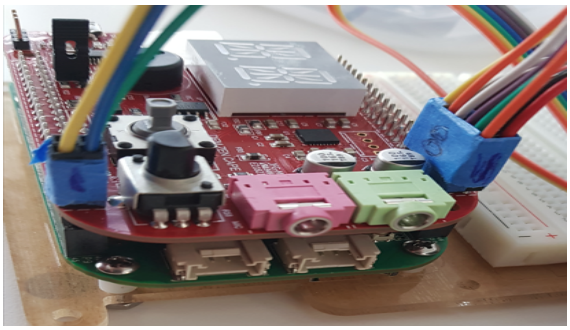
Figure 1: An example of pin mapping in chart and corresponding defined variables in C (Referred Adafruit official guide and Raymond Chan's guide)

Steps for connecting the wires:

1. Tape the LED panel's 4 ground(GND) pins together and connect to BBG's P9 pins

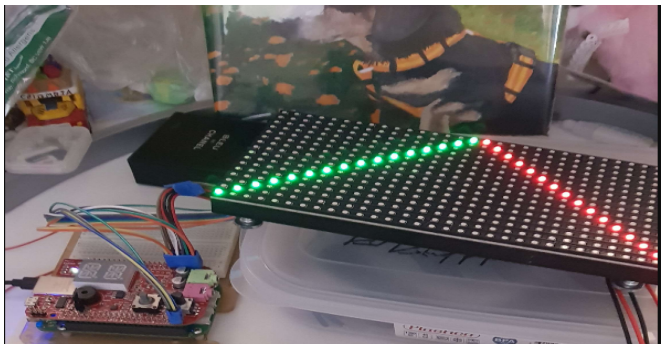
2. Tape the remaining 12 wires and connect to BBG's P8 pins.

3. Power your BBG with your host computer
4. Power your LED panel by connecting the power adapter

To test if you've connected the wires correctly, run test_LedMatrix.c, which was created by previous students. Upon running the executable, you should see a V-shape with one red and

one green side without excessive flickering. If you see the wrong pattern or color, double-check that the wires are plugged in tightly. You can reuse this as your LED matrix module and supply your own 2D array.
[GitHub - Montreal/BeagleBone-Green-Adafruit-16x32-LED-Matrix-Sample-Code]



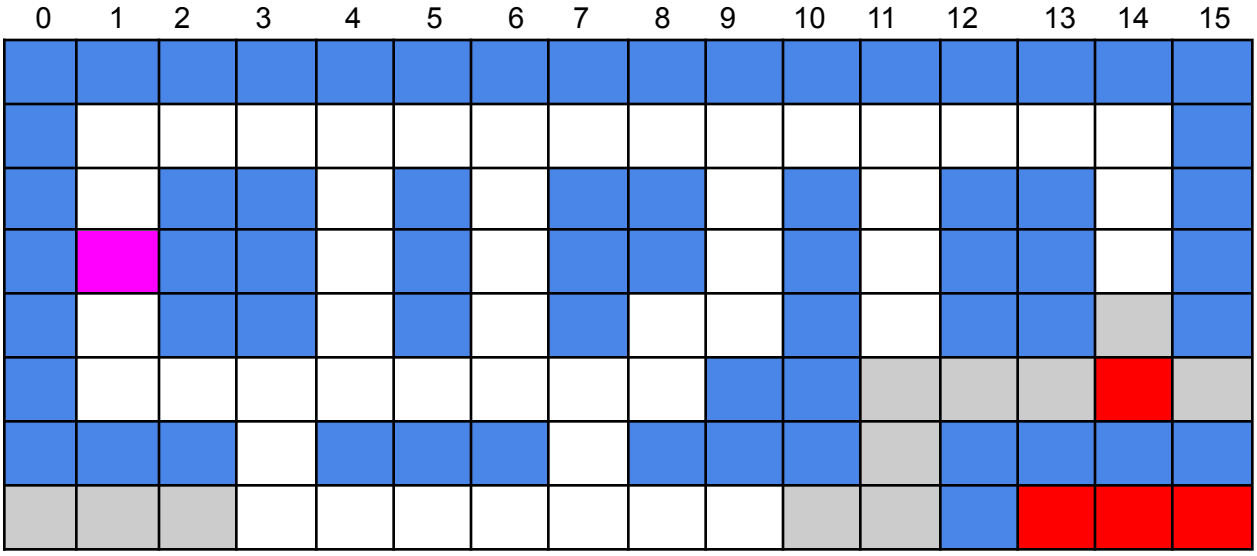
Tips:

1. You control the colour with 3 bits(R, B, G). Thus, you can generate 8(2^3) different colours(red, green, blue, pink, white, black, yellow, and light blue)
2. It is easier to manage the 16x32 2D array by dividing it into 4 segments. For example, our team created a game map for Pacman by initializing a 2D array in this manner.

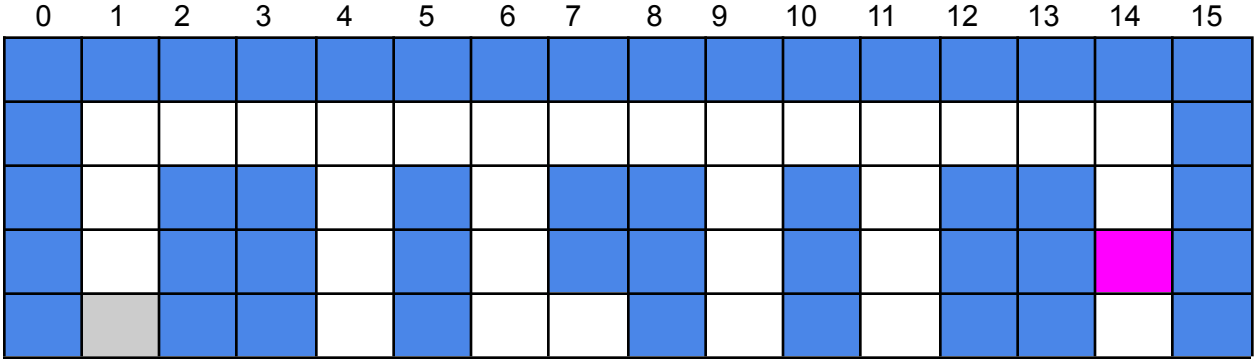
```
typedef enum
{
    WALL,
    PACMAN,
    DOT,
    POWER_DOT,
    GHOST,
    EMPTY,
    OUT_OF_BOUND
} TileType;

typedef struct
{
    TileType tileType;
    Color color;
} Tile;
 // Initializing map. ROW_SIZE = 16 COLUMN_SIZE = 32
    Tile mapTopLeft[ROW_SIZE / 2][COLUMN_SIZE / 2] = {
        {// trimmed},
        {// trimmed},
        {// trimmed},
        {// trimmed},
        {// trimmed},
        {// trimmed},
        {// trimmed},
```

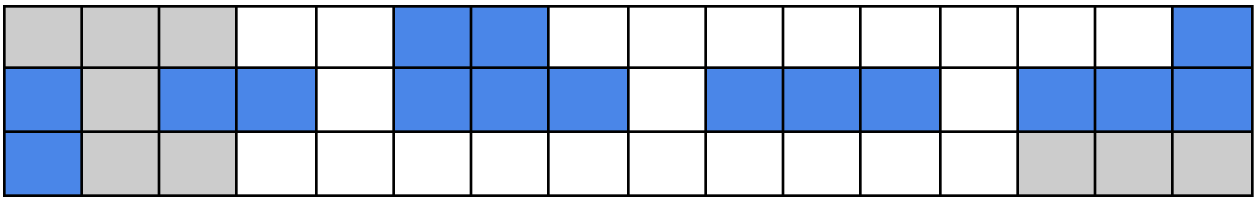
```
        {// trimmed}};

    Tile mapTopRight[ROW_SIZE / 2][COLUMN_SIZE / 2] = { // trimmed };

    Tile mapBottomLeft[ROW_SIZE / 2][COLUMN_SIZE / 2] = { // trimmed };

    Tile mapBottomRight[ROW_SIZE / 2][COLUMN_SIZE / 2] = { // trimmed };

    // Merge gameMap parts
    for(int i = 0;i<ROW_SIZE/2;i++){
        for(int j=0;j<COLUMN_SIZE/2;j++){
            gameMap[i][j]=mapTopLeft[i][j];
            gameMap[i][j+COLUMN_SIZE/2]=mapTopRight[i][j];
            gameMap[i+ROW_SIZE/2][j]=mapBottomLeft[i][j];
            gameMap[i+ROW_SIZE/2][j+COLUMN_SIZE/2]=mapBottomRight[i][j];
        }
```

Top Left

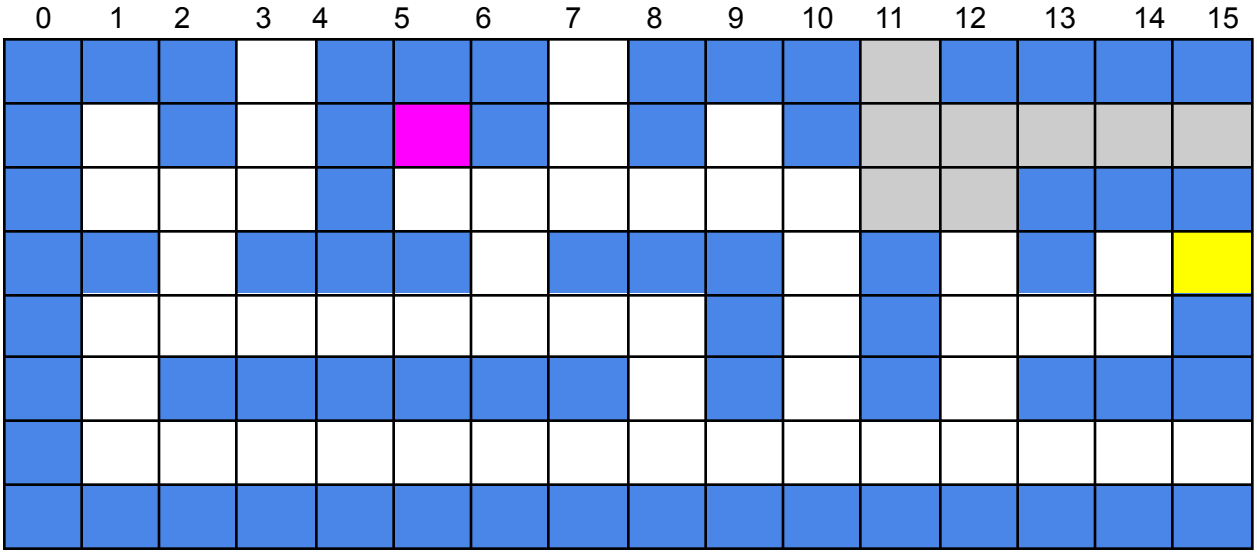


Top Right





Bottom Left



Bottom Right