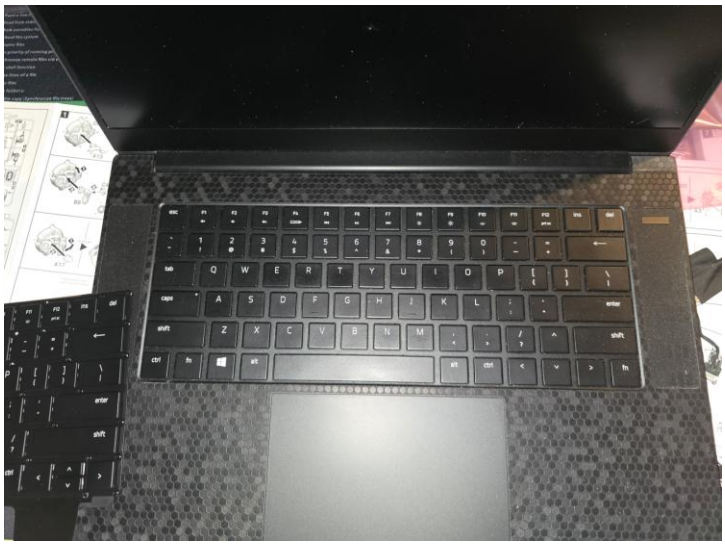
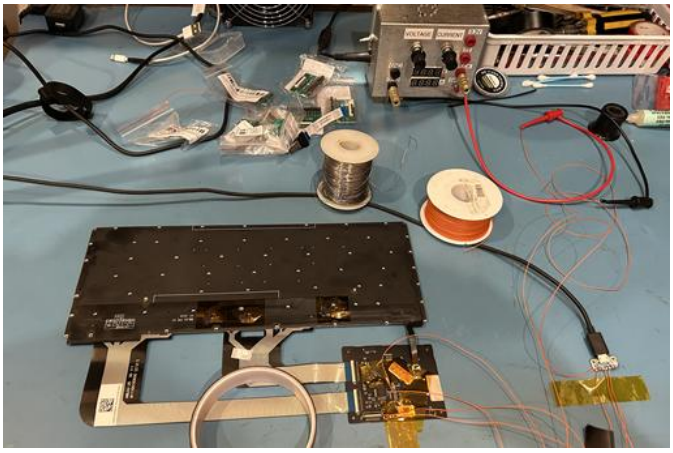
Razer Blade 15 Laptop Keyboard & Backlight Project



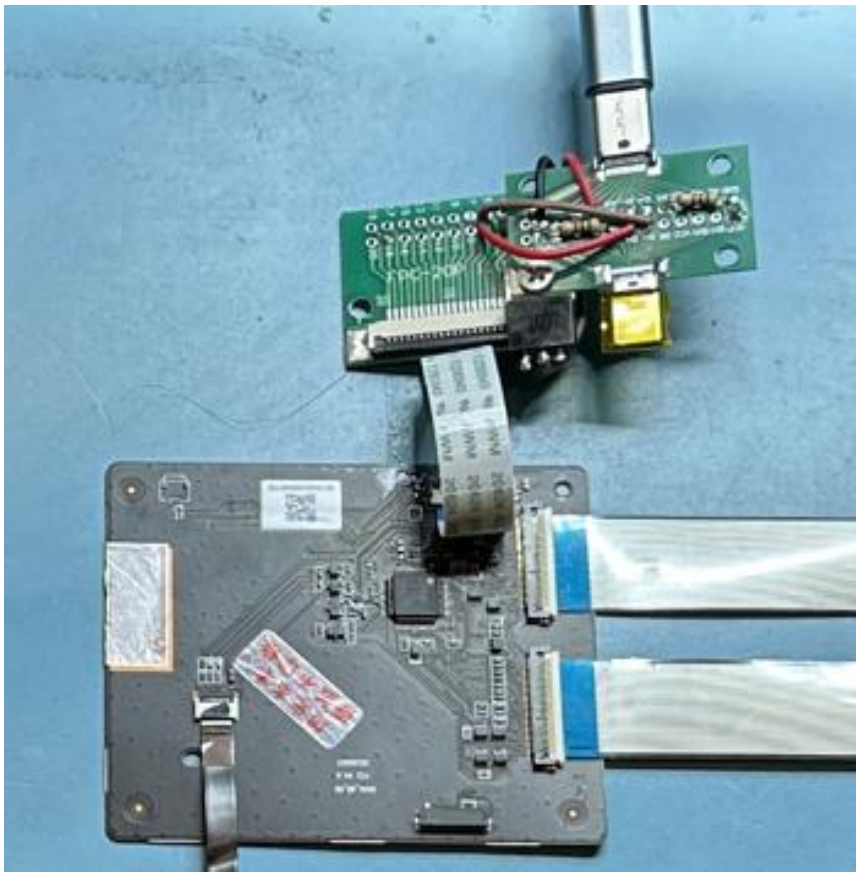
The Razer laptop keyboard shown above and on the left were made into USB keyboards using the original Razer Blade 15 controller board (shown below). This controller board uses an LPC11U37F ARM based IC to scan the keyboard with two 30 pin FPC connectors, control the backlight with an 8 pin FPC connector, and act as a USB HID device that previously went to the motherboard with a 20 pin FPC connector. All connectors have a 0.5mm pitch.

The LPC11U37F controller IC on the board is a 3.3 volt device. In order to connect this board to a USB bus from a PC, the 5 volts must be dropped to 3.3 volts. I tack soldered 4 wires to the controller board to connect 3.3 volts, ground, USB+, and USB-. This allowed me to temporarily connect to a PC and bench power supply for testing (see below).



All keys were correctly sent back over USB but the backlight would not turn on, even when Fn-F9 or Fn-F10 were pressed.

To use the 20 pin FPC cable instead of the jumpers, I built the adapter shown below.



The adapter assembly consists of a 20 pin FPC adapter board, a LM1117T-3.3 volt regulator and a USB 3 Type C breakout board. The USB signals on the 20-pin breakout board are: Data+ on FPC pin 6 and Data- on FPC pin 5. 3.3 volts from the voltage regulator goes to FPC pins 16, 17, or 18. They all get tied together on the controller board. Ground goes to FPC pins 4, 7, 10, 13, 15, or 19. These all get tied together on the controller board. Note that I used the pin numbers that are silk-screened on the FPC adapter board. The LM1117T-3.3 voltage regulator is in a TO-220 package and has ground on the left, 3.3 volts in the middle, and 5 volts on the right. A smaller regulator could have been used but this was what I had available. The USB C breakout board needed 5.1K resistors to ground on the A5 and B5 (CC1 and CC2) pins to tell the source it is a sink device with a default power level. Adafruit makes a smaller USB C breakout board for downstream connection that already includes the resistors.

The backlight is apparently controlled with special commands over the USB bus. Since I didn't have access to a motherboard, I couldn't capture these commands with a logic analyzer. I probed the backlight signals on the 8 pin backlight connector using an oscilloscope. I could tell it was a clock and data serial bus plus a logic signal driven by the controller chip that was always at 3.3 volts. I presumed this was an enable signal. I switched to a simple logic analyzer and determined the clock and data signals were I2C that communicated at addresses 0x3C, 0x3D, and 0x3F at a speed of about 400KHz. I later found these addresses correspond to the red, blue, and green LEDs. The power to the backlight was a noisy 2.7 volts. The pinout on the 8 pin FPC backlight connector is:

Ground = pins 1, 2

Enable = pin 3

Clock = pin 4

Data = pin 5

Power = pins 6, 7, 8

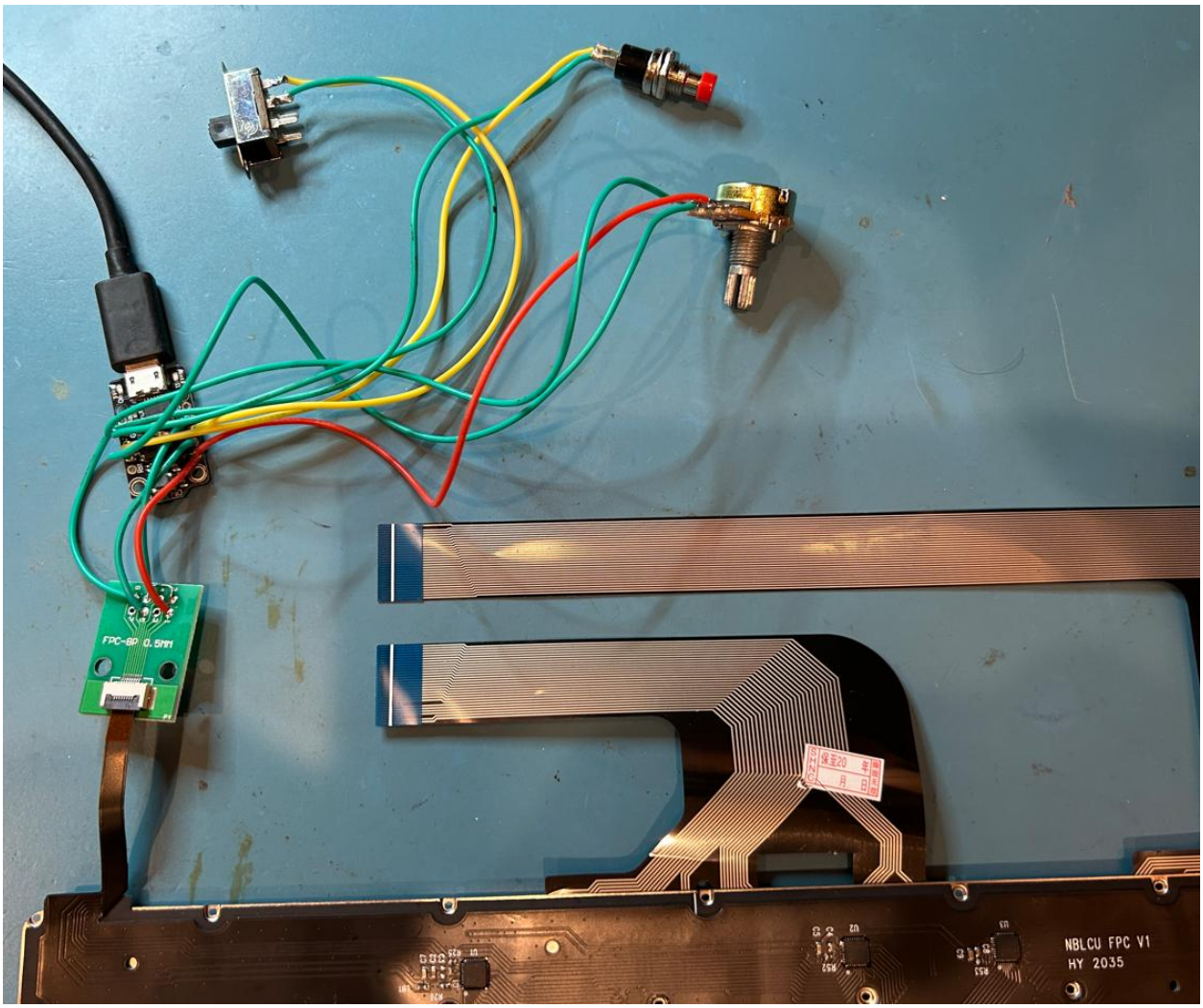
Note that pin 1 is on the left with a silk-screened dot (which is opposite of most FPC breakout boards).

The backlight circuit on the back of the keyboard consists of three 36 pin IC's, one for each LED color. I tested 2 keyboards and they had different ICs. One keyboard had three T1010 3235A chips and the other keyboard had three V1722 3235A chips. No data sheets could be found on the internet for either ICs.

Using the logic analyzer, I captured the startup commands given by the controller to the 3 backlight IC's. After startup, I noticed that all 3 chips were sent 28 bytes of zero's which

repeated every 10msec. To send my own signals, I wrote Arduino C code for an Adafruit Trinket M0 so it can be an I2C bus master. I used the default bus speed of 100KHz just to be safe. After sending the same startup commands, I experimented with sending 28 non-zero bytes and found these to be the LED brightness signals, 0xFF being full brightness. I also confirmed that the signal that was always driven high by the controller will disable the backlight if driven low. Since I didn't have an easy way to create 2.7 volts to power the backlight circuitry, I tried 3.3 volts and found it worked fine.

This picture shows an Adafruit Trinket M0 connected to an 8 pin FPC breakout board for the backlight cable. These boards are small enough that they can sit on top of the keyboard controller with electrical tape to keep them from shorting.



Note that the pin numbering that is silkscreened on the breakout board is backwards. The connections using the breakout board numbering are given below:

Ground = pins 8 and 7 They're tied together on the backlight so I only needed 1 wire to the Trinket ground pin.

Enable = pin 6 Since it's always high. I jumpered it to Trinket 3.3 volts to save a GPIO pin.

Clock = pin 5 Goes to Trinket GPIO 2. I added a 1.8K pull up resistor to 3.3 volts.

Data = pin 4 Goes to Trinket GPIO 0. I added a 1.8K pull up resistor to 3.3 volts.

Power = pins 3, 2, and 1 They're tied together on the backlight I only needed 1 wire to the Trinket 3.3 volt pin.

The Trinket M0 is programmed with a micro USB cable to a PC running the Arduino IDE. My code is called "Razer_Backlight.ino" and can be downloaded from my GitHub repo. Once programmed, the USB cable is only needed for 5 volt power. The USB cable could be eliminated if 5 volts and ground are wired directly to the LM1117T regulator. At full brightness, the 5 volt power draws only 240ma, well below what the regulator on the Trinket can supply.

In order to adjust the backlight, I added controls to the 3 remaining GPIO pins on the Trinket. I wired a 10K potentiometer with ground on one end, 3.3 volts on the other end, and the center is wired to ADC pin A3 on the Trinket. A push button switch is wired to GPIO 4 with the other side tied to ground. A slide switch is wired to GPIO 1 with the other side tied to ground.

The Arduino code is programmed to add a pull up to both switches so they are high unless engaged. After initializing the 3 backlight chips over the I2C bus, the code first checks to see if the push button switch is pressed. This switch causes pseudo random numbers to be sent for the red, green, and blue brightness levels and makes a colorful show. Next the code checks the slide switch to see which mode of operation is desired. If this switch is closed, the potentiometer voltage is read by the A to D converter and sent as the brightness value to all red, green, and blue LEDs (making them white). The user can adjust the LEDs from fully off to fully on. If the mode switch is off, there two ways the user can adjust the LEDs. Either push and release the push button switch multiple times until the random brightness and colors are acceptable, or send commands over the USB cable (assuming you didn't wire 5 volt power directly). The code uses the USB cable as a simple Com port at 9600 baud. The Arduino IDE has a serial Com window that allows the user to communicate to the Trinket and receive messages back. This Com port can also be created with Python code on a PC if you also install the PySerial module.

From the Arduino IDE Com window, type the following letters (and hit Enter) to change the brightness. Changes are in small steps so it takes multiple times to see the effect. If you make a mistake, the code will say "Invalid command" and list the possible commands shown below.

u = up brightness for all 3 colors

d = down brightness for all 3 colors

ur = up brightness for red

ub = up brightness for blue

ug = up brightness for green

dr = down brightness for red

db = down brightness for blue

dg = down brightness for green

mid = sets all colors at mid-scale (white)

off = turns all colors off

max = sets all colors at full brightness (white)

This picture shows full (white) brightness with the potentiometer.

This picture shows how the random number mode can give different colors and brightness. I had to push the button a few times to get the colors I liked.