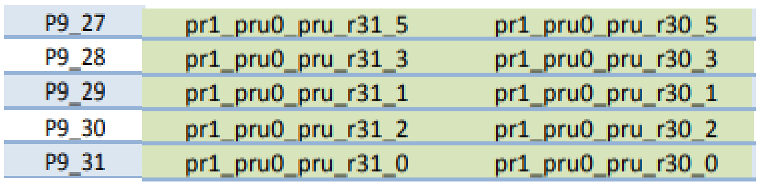
Adafruit NeoPixel Matrix Guide

This guide will be useful to drive an 8x8 NeoPixel RGB matrix on a Beagle Bone Green (should not be different from other version of BB or for larger matrices).

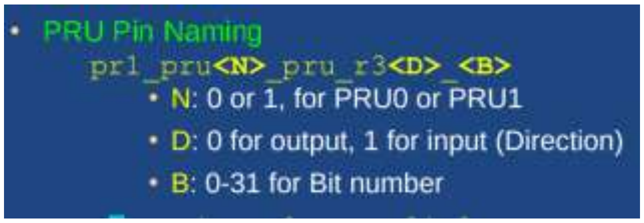
We have to use an external power source which is able to produce up to 3 Amps of current (we determined experimentally that 0.5A -1 A is good enough for most practical tasks, but driving the LED at its highest power, when all 64 pixels are turned white, can actually pull up to 3 Amps). Because a BBG can only provide 4-6 mA, it is not recommended to ever power the NeoPixel directly from BBG.

As for the voltage needed for NeoPixel, it has proved to work fine with a 3.3V power source, which matches the voltage provided by the BBG output pins and therefore the signal received by the DIN (data IN) wire should always be interpreted correctly. Therefore we can make it work without a lever shifter, that's recommended by Adafruit website. As always it is a good habit to place a resistor (around 1k is fine) between the BBG output pin and the DIN wire of the matrix.

As it turns out the easiest way to control the NeoPixel, and any PRU driven device in general, is by using the input and output registers: __R31 and __R30, that are predefined in internal PRU files. Here's a little snapshot of 5 of the most used pins (p9.27 through p9.31), and the corresponding bits of the PRU registers. The last digit of the code corresponds to the bit number of each pin. This is provided by Dr. Brian Fraser notes and guides.



P9_27	pr1_pru0_pru_r31_5	pr1_pru0_pru_r30_5
P9_28	pr1_pru0_pru_r31_3	pr1_pru0_pru_r30_3
P9_29	pr1_pru0_pru_r31_1	pr1_pru0_pru_r30_1
P9_30	pr1_pru0_pru_r31_2	pr1_pru0_pru_r30_2
P9_31	pr1_pru0_pru_r31_0	pr1_pru0_pru_r30_0



- PRU Pin Naming

  pr1_pru<N>_pru_r3<D>_<B>
  - N: 0 or 1, for PRU0 or PRU1
  - D: 0 for output, 1 for input (Direction)
  - B: 0-31 for Bit number

For the purpose of this guide we are going to use pin P9_27 which is associated with bit number 5 (offset from the least significant bit).

Here's a simple program to check to correct wiring (turn first led green). Before running the code don't forget to run "beaglebone> config-pin P9.27 pruout"

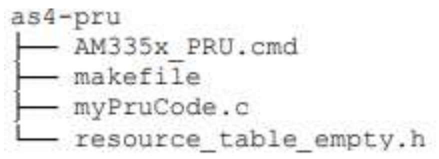
```
#include <stdint.h>
#include <stdio.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"

#define oneCyclesOn     700/5   // Stay on 700ns
#define oneCyclesOff    800/5
#define zeroCyclesOn    350/5
#define zeroCyclesOff   600/5
#define resetCycles     60000/5
#define DELAY_1_MS 200
#define LED_MASK (1 << 5)
volatile register uint32_t __R30;
volatile register uint32_t __R31;
void bit_on(void){
    __R30 |= LED_MASK;
    __delay_cycles(oneCyclesOn-1);
    __R30 &= ~LED_MASK;
    __delay_cycles(oneCyclesOff-2);
}
void bit_off(void){
    __R30 |= LED_MASK;
    __delay_cycles(zeroCyclesOn-1);
    __R30 &= ~LED_MASK;
    __delay_cycles(zeroCyclesOff-2);
}
void main(void)
{
    uint32_t green = 0xff0000; //green color code
    for(i=23; i>=0; i--) {
            if(green & (0x1<<i)) {
                bit_on();
            }
            else {
                bit_off();
            }
}
```

Or rather use these two functions two set any number of leds with a color of your choice (you can find color coded anywhere online, just don't forget to switch the order to: green, red, blue). Call TurnAllColor(0,64) at the end to clear the matrix.

```
void TurnAllColor(uint32_t custom_color, int num){
    int i, j;
    for(j=0; j<num; j++) {
        for(i=23; i>=0; i--) {
            if(custom_color & (0x1<<i)) {
                bit_on();
            }
            else {
                bit_off();
            }
        }
    }
}
```

All this code should go into the myPruCode.c file as shown in Dr. Brian Fraser's guide.
Everything else is also provided by Dr. Brian.



```
as4-pru
├── AM335x_PRU.cmd
├── makefile
├── myPruCode.c
└── resource_table_empty.h
```

You can make use of the code provided in NeoPixel-linux folder to create different patterns and send them to a PRU code via shared memory struct, using functions in led.c file. This is the shared struct currently used. Suggestion: use array of (int) to give each bit (each led) its own color, that would also require a more complicated linux high-level code.

```
typedef struct {
    int size; //to get the size of valid data points
    int mode; //to switch between modes of pru (mode 2 means read data from linux)
    int color; //  to apply this color for all 64 pixels
    bool doneExec;  //to signal to linux that pru is finished displaying
    bool dataReady;  // to signal pru that new data has arrived for display
    bool data[64];   // 64 bits either ON or OFF

} sharedMemStruct_t;
```