Easy PRU programming on BBG with Adafruit 8*8 NeoMatrix

Our guide will be more focused on running the Adafruit 8*8 NeoMatrix lights (or any other Adafruit NeoMatrix lights with chip) which requires PRU for timings. However this guides provides with the infrastructure setup such that the reader can code anything regarding PRU timing and interfacing with GPIO or PWM in C.

Index

1. What is PRU and why it is useful and powerful?

A PRU is a Programmable Read-Time Unit is embedded in the AM335x chip, which are two 32-bit 200MHz RISC cores. These cores are independent of the ARM which is responsible of running everything (such as .c or .cpp programmes) or anything else in User-space of BeagleBone. However PRU is isolated from the ARM so your PRU independent or ARM and that's why its PRU is **preferred for fast real-time operations**. PRU has its own 8KB of program memory and data-memory each. There are two PRU available to be used at a time. Due to abundance of technical details, this guide pertains

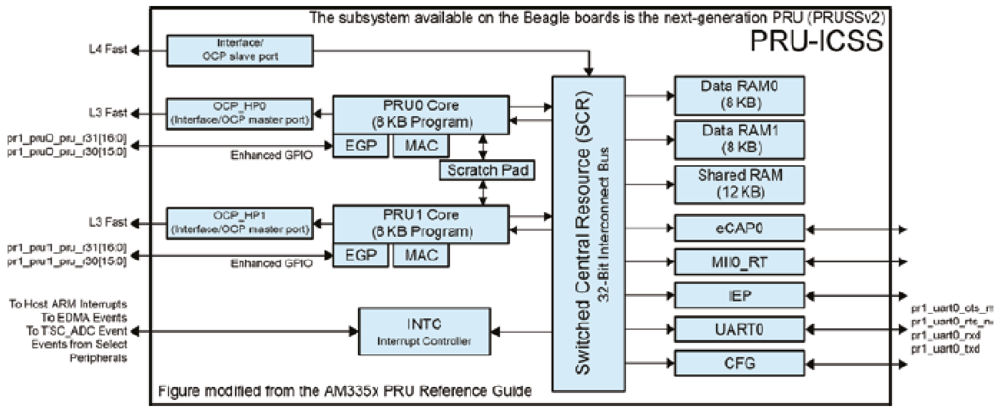


Figure 15-1: The PRU-ICSS architecture
Customized for the Beagle boards from an image that is courtesy of Texas Instruments

to a certain level of abstraction such that It is easy for course's student to interact PRU without any outside Library but still write student-interpretable code (in C).

In Derek's book on SFU Library Website or Amazon or Copy from Brian Fraser, there is a nice-introduction to PRU in chapter "Real-Time Interfacing with the PRU-ICSS" and also provides a sample program to run a PRU-driven LED.

A good use-case scenario is if you want to drive a external simple LED on a breadboard as well as a DC motor (which requires PWM). You can use two PRUs (PRUN=0 and PRUN=1) which will not interfere with each other (Diagram above.)

2. Basic Infrastructure Setup

Because PRU is not in ARM space, it will require its own dedicated compiler, assembler and linker. My guide is strongly based on PRUCookbook whose documentation helped us with getting up and running with Adafruit 8*8 Neomatrix lights (or any other Adafruit lights with WS2812 chip).

PRUCookbook Chap02 GitLink

There are some files which are core to the infrastructure such as:

- **Makefile** : This is our interface that does all the heavy lifting from starting PRU components on BBG, compiling our special C program, linking (lnkpru), compile (clpru) and stop PRU.
- **AM335x_PRU.cmd** : Linker file defines all the Register values which PRU needs to know. You can directly look at these registers in /sys/kernel/debug/remoteproc/remoteproc1 or /sys/kernel/debug/remoteproc/remoteproc2
- **Resource_table_empty.h :** Defines resource table for all PRU cores. Mainly the remoteproc will need this.
- **Setup.sh :** It populates some environmental variables such as model of beaglebone and configure pins accordingly. The env variables are used by Makefile.
- **Prugpio.h :** Definition of all GPIO Pins, USR registers (three LEDs embedded on BBG), Shared Mem (BBG). You can see multiple definitions of same pins because the pin mapping depends on the Processor chip(AM5729 vs AM335x (ours)).

We will not be actively changing all the files. Our main interaction will be with **Makefile**, **setup.sh** and the **.c file** you will program (we will show how later one). To Run your first PRU program, referring the code here .

BlinkLED.c is simple PRU program to power on/off a simple LED with PRU cycle delays. In the code :

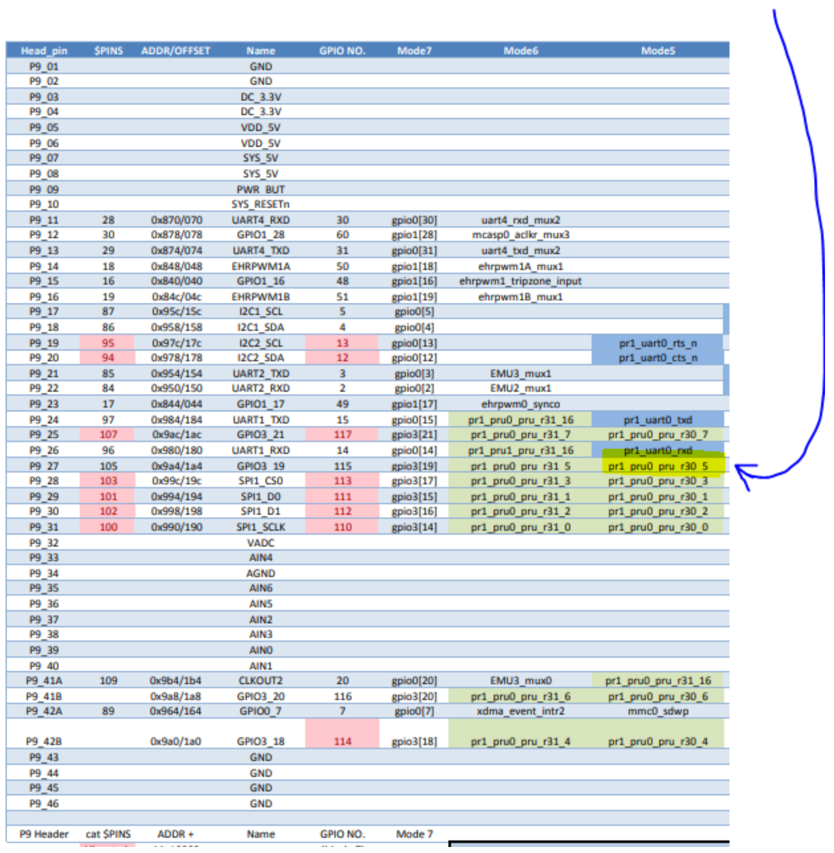
```
#include
<stdint.h>
             #include <pru_cfg.h>
             #include "resource_table_empty.h"
             volatile register uint32_t __R30;
             volatile register uint32_t __R31;
             void main(void)
             {
                    volatile uint32_t gpio;
                    // Clear SYSCFG[STANDBY_INIT] to enable OCP master port
                    CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;
                    // Use pru0_pru_r30_5 as an output i.e., 100000 or 0x0020
                    gpio = 0x0020;
                    // Infinite loop
                    while (1) {
                           __R30 ^= gpio;
                            // delay for 0.25s (one quarter second
                           __delay_cycles(50000000);
                    }
             }
```

Some important things :

- __R30 : This is a register which passes input to the (PRU 1) pru1_1
- __R31 : This is a register which received outputs from (PRU 1) pru1_1

So, basically, if you want to write to PRU (__R30), you'll use the expression

__R30 ^= gpio where gpio can be any GPIO or USR as defined in prugpio.h. You can also define specific GPIO pin locally in BlinkLED.c and not take from prugpio.h. In this example, gpio=0x0020 refers to **pru0_pru_r30_5** which is GPIO pin P9_27 (gpio#115 in /sys/class/gpio). Make sure you set the P9_27 to pruout through the command : **config-pin P9_27 pruout.**



Head_pin	$PINS	ADDR/OFFSET	Name	GPIO NO.	Mode7	Mode6	Mode5
P9_01			GND				
P9_02			GND				
P9_03			DC_3.3V				
P9_04			DC_3.3V				
P9_05			VDD_5V				
P9_06			VDD_5V				
P9_07			SYS_5V				
P9_08			SYS_5V				
P9_09			PWR_BUT				
P9_10			SYS_RESETn				
P9_11	28	0x870/070	UART4_RXD	30	gpio0[30]	uart4_rxd_mux2	
P9_12	30	0x878/078	GPIO1_28	60	gpio1[28]	mcasp0_aclkr_mux3	
P9_13	29	0x874/074	UART4_TXD	31	gpio0[31]	uart4_txd_mux2	
P9_14	18	0x848/048	EHRPWM1A	50	gpio1[18]	ehrpwm1A_mux1	
P9_15	16	0x840/040	GPIO1_16	48	gpio1[16]	ehrpwm1_tripzone_input	
P9_16	19	0x84c/04c	EHRPWM1B	51	gpio1[19]	ehrpwm1B_mux1	
P9_17	87	0x95c/15c	I2C1_SCL	5	gpio0[5]		
P9_18	86	0x958/158	I2C1_SDA	4	gpio0[4]		
P9_19	95	0x97c/17c	I2C2_SCL	13	gpio0[13]		pr1_uart0_rts_n
P9_20	94	0x978/178	I2C2_SDA	12	gpio0[12]		pr1_uart0_cts_n
P9_21	85	0x954/154	UART2_TXD	3	gpio0[3]	EMU3_mux1	
P9_22	84	0x950/150	UART2_RXD	2	gpio0[2]	EMU2_mux1	
P9_23	17	0x844/044	GPIO1_17	49	gpio1[17]	ehrpwm0_synco	
P9_24	97	0x984/184	UART1_TXD	15	gpio0[15]	pr1_pru0_pru_r31_16	pr1_uart0_txd
P9_25	107	0x9ac/1ac	GPIO3_21	117	gpio3[21]	pr1_pru0_pru_r31_7	pr1_pru0_pru_r30_7
P9_26	96	0x980/180	UART1_RXD	14	gpio0[14]	pr1_pru1_pru_r31_16	pr1_uart0_rxd
P9_27	105	0x9a4/1a4	GPIO3_19	115	gpio3[19]	pr1_pru0_pru_r31_5	pr1_pru0_pru_r30_5
P9_28	103	0x99c/19c	SPI1_CS0	113	gpio3[17]	pr1_pru0_pru_r31_3	pr1_pru0_pru_r30_3
P9_29	101	0x994/194	SPI1_D0	111	gpio3[15]	pr1_pru0_pru_r31_1	pr1_pru0_pru_r30_1
P9_30	102	0x998/198	SPI1_D1	112	gpio3[16]	pr1_pru0_pru_r31_2	pr1_pru0_pru_r30_2
P9_31	100	0x990/190	SPI1_SCLK	110	gpio3[14]	pr1_pru0_pru_r31_0	pr1_pru0_pru_r30_0
P9_32			VADC				
P9_33			AIN4				
P9_34			AGND				
P9_35			AIN6				
P9_36			AIN5				
P9_37			AIN2				
P9_38			AIN3				
P9_39			AIN0				
P9_40			AIN1				
P9_41A	109	0x9b4/1b4	CLKOUT2	20	gpio0[20]	EMU3_mux0	pr1_pru0_pru_r31_16
P9_41B		0x9a8/1a8	GPIO3_20	116	gpio3[20]	pr1_pru0_pru_r31_6	pr1_pru0_pru_r30_6
P9_42A	89	0x964/164	GPIO0_7	7	gpio0[7]	xdma_event_intr2	mmc0_sdwp
P9_42B		0x9a0/1a0	GPIO3_18	114	gpio3[18]	pr1_pru0_pru_r31_4	pr1_pru0_pru_r30_4
P9_43			GND				
P9_44			GND				
P9_45			GND				
P9_46			GND				
P9 Header	cat $PINS	ADDR +	Name	GPIO NO.	Mode 7		

3. Choosing Pins :

You can refer the BBG GPIO Pin layout on CMPT433 websites for P8 Headers and P9 Headers.

If you want to output to PRU, choose PRU number (0 or 1) first. Then search for pr1_pru**A**_pru_3**B**_**C** (**A**=PRU# either 0 or 1 ,**B** = 0/1 (output or input), **C =** GPIO#). For instance, pr1_pru0_pru_r30_5 is GPIO Pin **P9_27** running **PRU 0** to **output.** Mode 5 in the chart above have output pins while Mode4 have input. Again If you want to set a GPIO to PRU output or input, you will configure them :

```
config-pin pin# pruin
```

```
config-pin pin# pruout
```

More info about this stage can be found here in PRUCookbook tutorial

4. Interacting with PRU in real-time outside PRU Space

For instance, you were able to run PRU through command : make PRUN=0 TARGET=BlinkLED , and turn it off make PRUN=0 stop. Next thing is how you are going to interact with the BlinkLED program?. In normal .c you can interact via File I/O, stdin/stdou, Pipes etc. Because your BlinkLED is running in PRU Space, you can't use traditional ways to interact with your code from outside.

```
@sudo cp $(GEN_DIR)/$(TARGET).out /lib/firmware/am335x-pru$(PRUN)-fw
```

This snippet from Makefile copies your .c code to special directory /lib/firmware/am335x-pruX –fw, your blinkLED.c is not your regular C code. For instance, you can't use printf. (Yes you have to be careful during testing and many other libraries.). The include_path tells you what libraries you can use in your PRU C code.

That's why we will use **prmsg kernel driver**. The Example code can be found here : under section " 1.14 Controlling NeoPixels through a Kernel Driver"

The workflow is this :

1. `sudo chmod 666 /dev/rpmsg_pru30`
2. `echo "my message" > /dev/rpmsg_pru30`

Basically, the code line here in neo4.c (from PRUCookbook link)

```
while (pru_rpmsg_receive(&transport, &src, &dst, payload, &len) == PRU_RPMSG_SUCCESS)
```

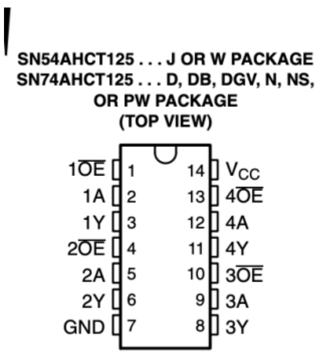
In Variable payload, you will receive the message written to /dev/rpmsg_pru30 .In the Example code, you can see the parsing techniques (strtol with strchr functions).

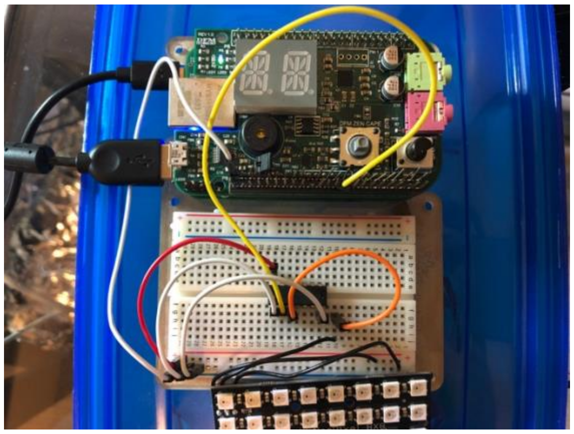
This means you can make another .c program or shell programs that can interact with your PRU. Again to clarify, PRU .c program that have actual code is stored inside /lib/firmware… , you can write to /dev/rpmsg_pru30 to pass message from ARM to PRU and in the /lib/firmware … c code, your message will be stored inside payload variable.

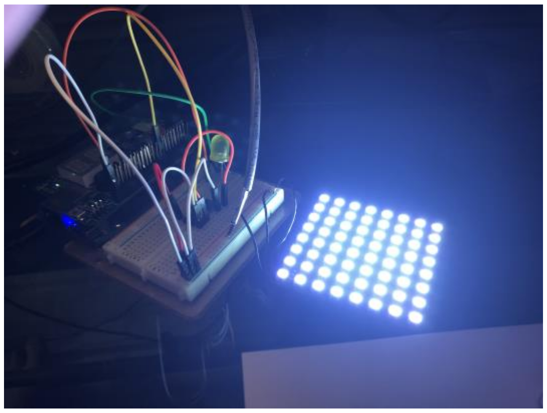
5. Adafruit 8*8 NeoMatrix with PRU

The example in PRUCookbook is a great resource for running PRU driven application in BeagleBone Green. However, understanding how PRU works vs actually running PRU code are two different things. Another PRU guide in cmpt433 goes through low-level PRU command line tools which are not useful for running out .c type code.

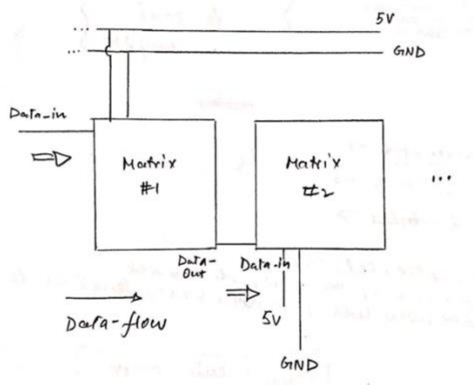
We followed the base circuit diagram same as RaspberryPi here with some changes. Adafruit neo products require 5V so we use Level-shifters and its data-sheet will explain







The Neomatrix has three wires: Data-in, Ground and Power. The Ground and Power are self-explanatory. For Data-in it goes through level-shifter (Orange wire to 1Y of level-shifter). In the Image above, you also have to connect 5V wires into + and – to make circuit complete (Image on left)



BONUS : It is possible to join multiple NeoMatrix together for more fun. You have to solder one more wire for Data-out (opposite side of Neomatrix) and that connects to Data-in of next Neomatrix. We have not done and leave it to the reader to experiment with this.

For our 8*8 Adafruit lights (https://www.adafruit.com/product/1487) to run on BeagleBoneGreen, you can take the basic infrastructure code(Makefile,resource-table,setup.sh etc..) from either PRUCookbook example or Derek's BeagleBone book's example code and start changing to your needs. I will quickly go over changes I made to run NeoMatrix:

1. Makefile :
    a. Some permissions problems. Add sudo to lines where there's command such as `sudo cp $(GEN_DIR)/$(TARGET).out /lib/firmware/$(CHIP)-pru$(PRUN)-fw`
    b. Support Files : Change directories for external files such as resource_table or AM335x_pru.cmd. Its preferred to keep everything in same folder for simplicity.
    c. Compile your own normal .c program : Though Makefile is for compiling PRU C programs using it's own compiler and linker. However you can compile your own code. For instance

```
neomatrix_interface :
	$(CC_C_INTERFACE) $(CFLAGS_INTERFACE) $@.c $^ -o
$(NEOMATRIX_OUTOUTDIR)/$@ -lpthread -lm
```

Inside Makefile also compiles my neomatrix_interface.c which writes to /dev/rpmsg_pru30.

This is how my default make looks like :

```
all: setup neomatrix_interface install start
```

Then inside my PRU C function I wrote with neo4.c(in PRUCookbook Chapt-5) as a base program :

```
void TurnAllGreen(void){

    uint32_t custom_color = 0x0f0000; // blue color
    uint32_t color[64] ;

    int i, j, k;
    for(k=0;k<64;k++){
        color[k] = custom_color;
    }


    for(j=0; j<64; j++) {
        for(i=23; i>=0; i--) {
            // logic to enable which leds to lit up.
            if(color[j] & (0x1<<i)) {
                bit_on();
            }
            else {
                bit_off();
            }

        }
    }
}
void bit_on(void){
    __R30 |= gpio;      // Set the GPIO pin to 1
    __delay_cycles(oneCyclesOn-1);
    __R30 &= ~(gpio);    // Clear the GPIO pin
    __delay_cycles(oneCyclesOff-2);
}
void bit_off(void){
    __R30 |= gpio;       // Set the GPIO pin to 1
    __delay_cycles(zeroCyclesOn-1);
    __R30 &= ~(gpio);    // Clear the GPIO pin
    __delay_cycles(zeroCyclesOff-2);
}
```

It is a sample program to turn on all LEDs green. You can call TurnAllGreen(void) inside PRU C code's main() to evoke. Originally, we have to write many other functions to simulate AudioVisualizer driven by Audio.

Also make the Adafruit Lights accepts GRB values. That means 0xff0000 = Pure Green , 0x00ff00 = Pure Red and 0x0000ff = Pure Blue. This requirement can be found in WS2812 data-sheet.

Note : If you have other variants of Adafruit Lights which still uses WS2812, this guide will be the same. However you have to change some variables above in TurnAllGreen(void) like , # of LEDS (other # than 64)

6. Conclusion

We tried to cover the important big stuff, however many other details have been omitted from the guide because of length restrictions. I would suggest to approach it this way:

1. Replicate PRUCookbook code or BlinkLED example from Derek Github on your BBG to ensure you understand and run Basic PRU program
2. Turn on/off simple LED Circuit (resistor, LED diode with + from gpio and – to ground) or USR LEDs on BBG.
3. Figure out wiring for your Adafruit Neo product.
4. Simple Turn on/off program for your Adafruit (like TurnAllGreen)
5. Then write your custom code in PRU C code or interact via Kernel Driver for more sophisticated programs.