

Published on *Analog Digital Lab:

electronics articles and tutorials* (<http://10.240.11.2>)

[Home](#) > [Articles](#) > [Beaglebone/Pi](#) > Introduction to BeagleBone Black PRU-ICSS

posted by [maintenance](#) on Fri, 10/04/2013 - 00:21

BeagleBone Black: Introduction to PRU-ICSS

Introduction to BeagleBone Black(BBB) PRU-ICSS

Authors: Haolin Li , Joris Van Kerrebrouck

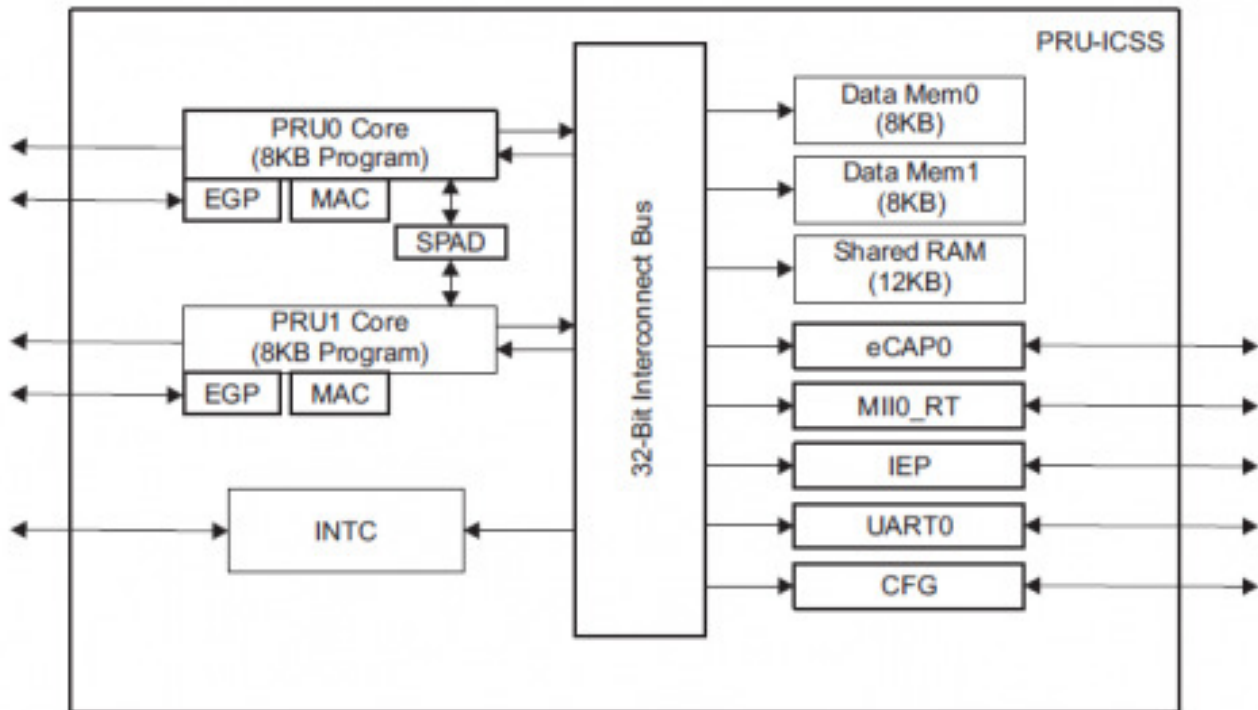
Linux 3.8 kernel

OS: Angstrom

Introduction

The Programmable Real-Time Unit and Industrial Communication Subsystem (PRU-ICSS) consists of dual 32-bit RISC cores (Programmable Real-Time Units, or PRUs), shared, data, and instruction memories, internal peripheral modules, and an interrupt controller (INTC). The programmable nature of the PRUs, along with their access to pins and events, provide flexibility in implementing custom peripheral interfaces, fast real-time responses, power saving techniques, specialized data handling and DMA operations, and in offloading tasks from the other processor cores of the system-on-chip (SoC).

The BeagleBone Black's TI chip ([XAM3359AZCZ revision 2](#)) has two PRUs(PRU0 and PRU1).The PRU cores are programmed with a small, deterministic instruction set. Each PRU can operate independently or in coordination with each other and can also work in coordination with the device-level host CPU. This interaction between processors is determined by the nature of the firmware loaded into the PRU's instruction memory.



Compared to microcontroller like PIC32 the additional PRU cores run at a far higher speed (200MHz) which in many cases means that external logic devices, CPLDs or FPGAs may not be necessary. This high speed feature makes the PRU very suitable for applications where high speed is a must like:

interfacing to a fast ADC (e.g. analog capture),
 CCD or a CMOS camera
 analog video generation (video encoder)
 parallel communication

or some general applications like:
 PWM or other custom or non-standard protocols
 motor control
 LED or LCD display
 ...

Getting started

1. download the file `am335x_pru_package-master.zip` from the [github](#) and save it onto the BBB and unzip to a folder.

personally, we changed the folder name to `am335x_pru_package`

```
mv am335x_pru_package-master am335x_pru_package
```

or directly use `git clone git://github.com/beagleboard/am335x_pru_package.git`

2. export `CROSS_COMPILE=`

if you use `CROSS_COMPILE=""`, there will be an error :

```
arm-arago-linux-gnueabi-gcc -I. -Wall -I../include -c -g -O0 -D__DEBUG -o debug/prussdrv.o  
prussdrv.c
```

```
make: arm-arago-linux-gnueabi-gcc: Command not found
```

```
make: *** [debug/prussdrv.o] Error 127
```

3. go to pru_sw/app_loader/interface

4. type: make

5. then go to pru_sw/utlis and type cd pasm_source

6. type source ./linuxbuild

if only type ./linuxbuild, it will cause an error

```
-sh: ./linuxbuild: Permission denied
```

7. in the directory /am335x_pru_package/pru_sw/utlis : mv pasm pasm_2

later you will see in the Makefile there exists pasm_2 instead of pasm

8. go to pru_sw/example_apps

9. open the Makefile, nano Makefile, you will see

```
CROSS_COMPILE?=arm-arago-linux-gnueabi-  
LIBDIR_APP_LOADER?=../app_loader/lib  
INCDIR_APP_LOADER?=../app_loader/include  
BINDIR_APPLICATIONS?=../bin  
BINDIR_FW?=bin  
  
PASM?=../utlis/pasm_2
```

that's why we changed pasm to pasm_2 in step 7.

10. close the Makefile and type make clean

11. type make

```

done : \
mv *.bin bin

PRU Assembler Version 0.84
Copyright (C) 2005-2013 by Texas Instruments Inc.

Pass 2 : 0 Error(s), 0 Warning(s)

Writing Code Image of 16 word(s)

PRU Assembler Version 0.84
Copyright (C) 2005-2013 by Texas Instruments Inc.

Pass 2 : 0 Error(s), 0 Warning(s)

Writing Code Image of 14 word(s)

PRU Assembler Version 0.84
Copyright (C) 2005-2013 by Texas Instruments Inc.

Pass 2 : 0 Error(s), 0 Warning(s)

Writing Code Image of 16 word(s)

PRU Assembler Version 0.84
Copyright (C) 2005-2013 by Texas Instruments Inc.

Pass 2 : 0 Error(s), 0 Warning(s)

Writing Code Image of 14 word(s)

root@BBB:~/am335x_pru_package/pru_sw/example_apps#

```

12. copy the .bin file in the example_apps/bin folder to the example_apps/xxx/obj folder (You will have to move it manually)

go to am335x_pru_package/pru_sw/example_apps/bin and type

```
cp PRU_memAccess_DDR_PRUsharedRAM.bin ../PRU_memAccess_DDR_PRUsharedRAM
/obj/PRU_memAccess_DDR_PRUsharedRAM.bin
```

13. make an executable file

```
cd PRU_memAccess_DDR_PRUsharedRAM/obj
```

```
gcc PRU_memAccess_DDR_PRUsharedRAM.o -L../lib -lprussdrv -lpthread -o
mytest.out
```

14. With both the executable (mytest.out) and the .bin file in the same folder, execute ./mytest.out

if you get the following text, then you need to make some modification.

INFO: Starting PRU_memAccess_DDR_PRUsharedRAM example.

```
prussdrv_open open failed
```

Solution:

```
cd /boot
cp am335x-boneblack.dtb am335x-boneblack.dtb_orig
dtc -I dtb -O dts am335x-boneblack.dtb > am335x-boneblack.dts_orig
```

```
nano am335x-boneblack.dts_orig
```

Search for pruss@4a300000 and then under it change

status = "disabled";

to

status = "okay";

```
dtc -I dts -O dtb am335x-boneblack.dts_orig > am335x-boneblack.dtb
```

```
reboot
```

This solution is not needed when we use a "dts fragment file" also known as ".dtbo" method to enable the PRU (covered in the example 1).

15. go to am335x_pru_package/pru_sw/example_apps

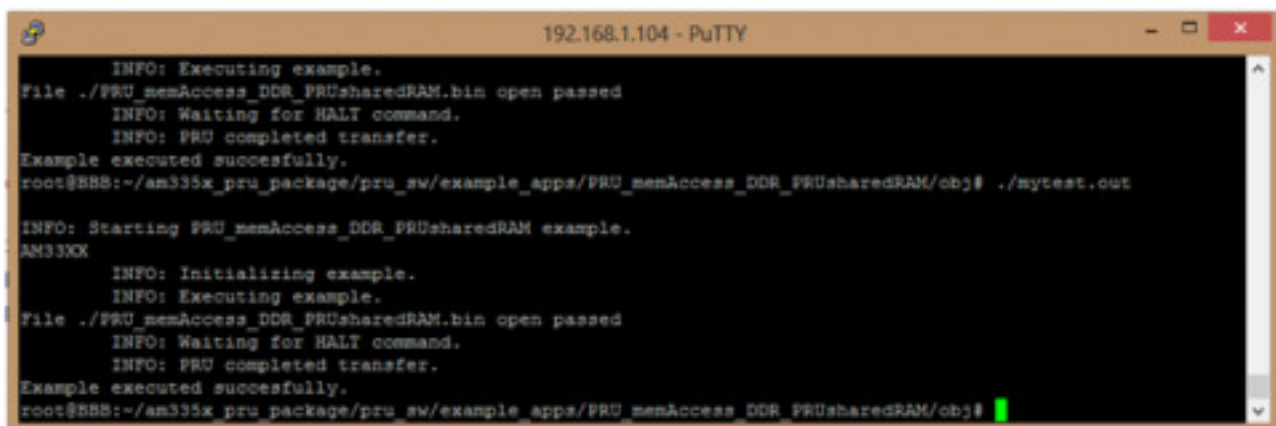
```
export CROSS_COMPILE=
```

```
make clean
```

```
make
```

```
repeat step 12-13
```

```
execute ./mytest.out
```



```
192.168.1.104 - PuTTY
INFO: Executing example.
File ./PRU_memAccess_DDR_PRUsharedRAM.bin open passed
INFO: Waiting for HALT command.
INFO: PRU completed transfer.
Example executed successfully.
root@BBB:~/am335x_pru_package/pru_sw/example_apps/PRU_memAccess_DDR_PRUsharedRAM/obj# ./mytest.out

INFO: Starting PRU_memAccess_DDR_PRUsharedRAM example.
AM33XX
INFO: Initializing example.
INFO: Executing example.
File ./PRU_memAccess_DDR_PRUsharedRAM.bin open passed
INFO: Waiting for HALT command.
INFO: PRU completed transfer.
Example executed successfully.
root@BBB:~/am335x_pru_package/pru_sw/example_apps/PRU_memAccess_DDR_PRUsharedRAM/obj#
```

The PRU applications contains

(a) the assembler code .p file that gets assembled into .bin file with the help of PASM.

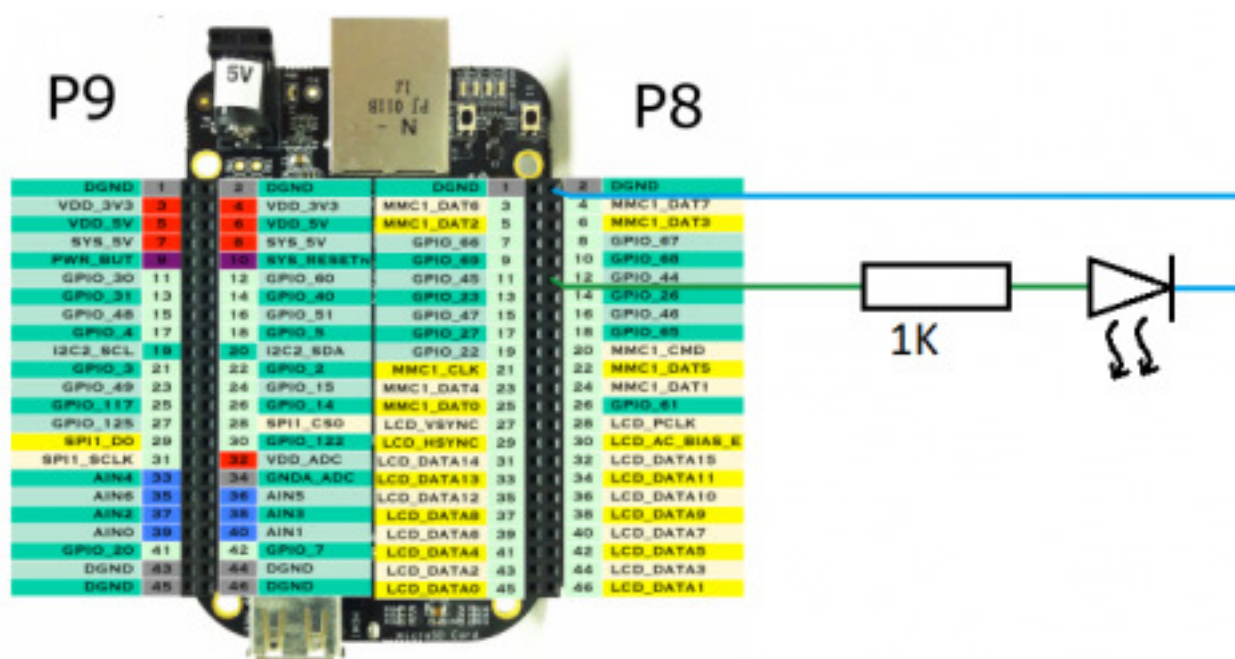
.bin file is located in `am335x_pru_package/pru_sw/example_apps/bin`

(b) c code running on the main processor, located in `am335x_pru_package/pru_sw/example_apps`

PASM is a command-line-driven assembler for the programmable real-time execution unit (PRU) of the programmable real-time unit subsystem (PRUSS). It is designed to build single executable images using a flexible source code syntax and a variety of output options. PASM is available for Windows and Linux.

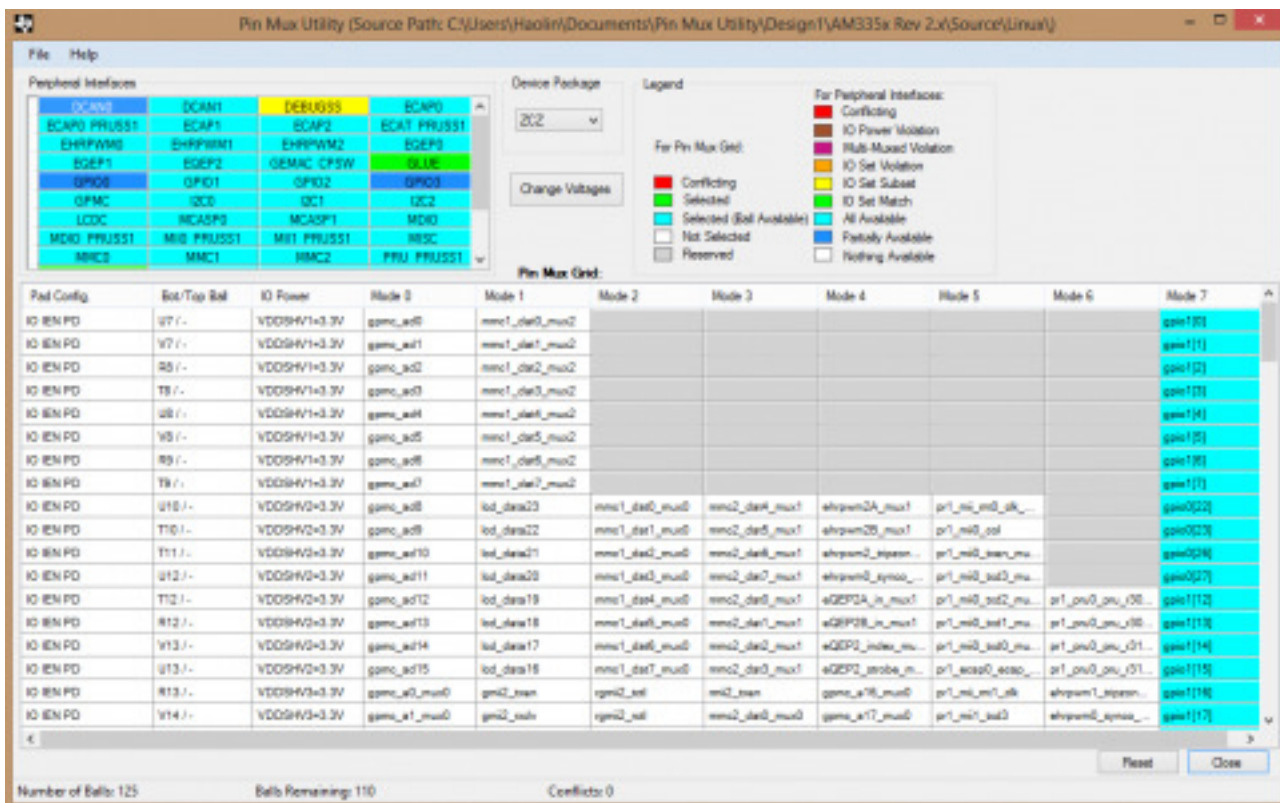
more details see *am335xPruReferenceGuide.pdf-P36* which is located in the zip file you've downloaded.

Example 1 - PRU0 blinking LED



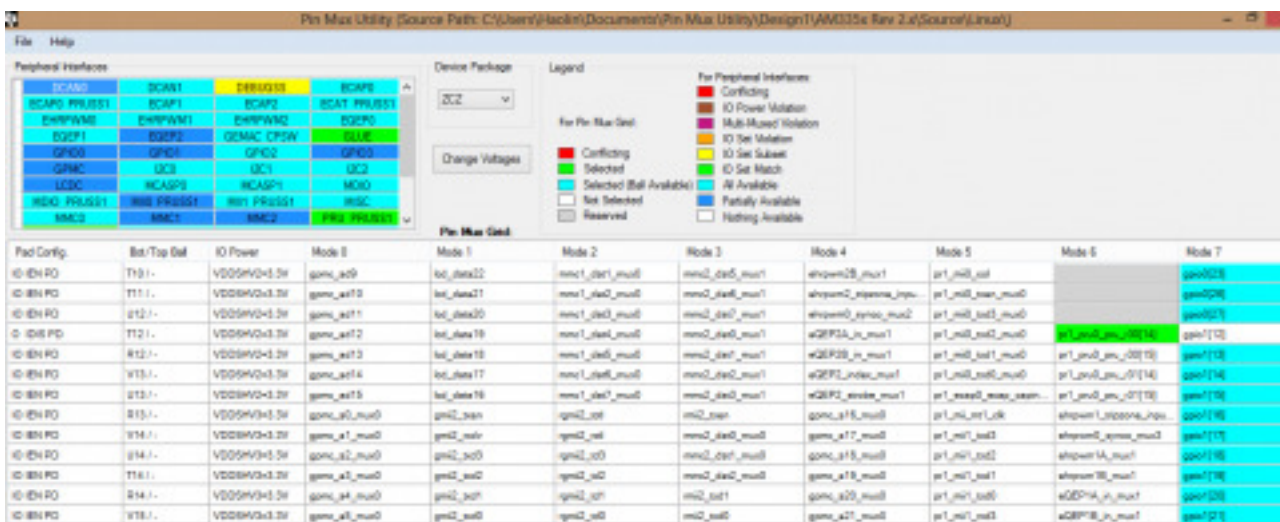
Step 1: configure pinout

we used `pinmux` utility to configure the in-/output pins. `PinMuxUtility_02_05_02_00.zip` is available from this [link](#). Launch the `PinMuxUtility` and select AM335x Rev 2X, ZCZ and IO Power Connections : 3.3V respectively.



There are 7 modes which the pins can be configured to. For this example we will configure the `pr1_pru0_pru_r30[14]` in mode 6 and as output pin. Double click the cell of `pr1_pru0_pru_r30[14]` will change to green.

`pr1_pru0_pru_r30[14]` corresponds to GPIO1[12], according to this [pinout table](#), it is P8.12 (pin 12 on connector P8)



Click then File->Save-> Source file ->Linux

There are two files named pinmux and mux.

Determine the address:

Open the pinmux file and you will see this line in MODE6:

```
MUX_VAL(CONTROL_PADCONF_GPMC_AD12, (IDIS | PD | MODE6 )) /* pr1_pru0_pru_r30[14] */^
```

Open the mux file and search CONTROL_PADCONF_GPMC_AD12, you will find

```
#define CONTROL_PADCONF_GPMC_AD12          0x0830
```

The 0x0830 is the offset with respect to the Control Module address AM335X_CTRL_BASE. according to page 158 of the tech ref manual and this [link](#).

The **AM335X_CTRL_BASE=0x44E10000** (also defined in the *pru.hp* see files for download, #define AM335X_CTRL_BASE 0x44E10000)

```
#define MUX_VAL(OFFSET,VALUE)\
writel((VALUE), AM335X_CTRL_BASE + (OFFSET));

#endif
```

So ordinarily we need to add the 0x830 offset to it, to get **0x44E10830**. However with the Device tree, according to this [link](#) when using the device tree the address to use is relative to the base address of the pin gpmc_ad0. That base address is **0x44e10800**. So, the number that will be used in the device tree is 0x30.

Determine the value:

You need to set this register to the value (IDIS | PD | MODE6).

at the top of MUX file:

```
/*

* MODE0 - Mux Mode 0

* MODE1 - Mux Mode 1

* MODE2 - Mux Mode 2

* MODE3 - Mux Mode 3

* MODE4 - Mux Mode 4

* MODE5 - Mux Mode 5

* MODE6 - Mux Mode 6
```


- * MODE7 - Mux Mode 7
- * IDIS - Receiver disabled
- * IEN - Receiver enabled
- * PD - Internal pull-down
- * PU - Internal pull-up
- * OFF - Internal pull disabled
- */

```
#define MODE0 0
```

```
#define MODE1 1
```

```
#define MODE2 2
```

```
#define MODE3 3
```

```
#define MODE4 4
```

```
#define MODE5 5
```

```
#define MODE6 6
```

```
#define MODE7 7
```

```
#define IDIS (0 << 5)
```

```
#define IEN (1 << 5)
```

```
#define PD (0 << 3)
```

```
#define PU (2 << 3)
```

```
#define OFF (1 << 3)
```

IDIS(0<<5): '0' at bit 5

PD(0<<3) : '0' at bit 3

MODE 6: '1' at bit 2 and bit 1

So (IDIS | PD | MODE6) gives us 0x06.

Step 2: configure Linux device tree

The Device Tree (DT), and Device Tree Overlay are a way to describe hardware in a system. It describes which pins, how they should be muxed, and which driver to use.

The original BeagleBone didn't use the DT, but the recently released BeagleBone Black was released with the DT and is now using the 3.8 Linux Kernel.

Details see [introduction-to-the-beaglebone-black-device-tree](#)

```
cd /lib/firmware
```

```
nano BB-BONE-PRU-00A0.dts
```

Copy the following code(the code is written by [shabaz](#) and is used here as a starting point.)

```
/*
 * pru dts file BB-BONE-PRU-00A0.dts
 */
/dts-v1/;
/plugin/

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";

    /* identification */
    part-number = "BB-BONE-PRU";
    version = "00A0";
    exclusive-use =
        "P8.12";

    fragment@0 {
        target = <&am33xx_pinmux>;
        __overlay__ {
            mygpio: pinmux_mygpio{
                pinctrl-single,pins = <
                    0x30 0x06
                >;
            };
        };
    };
};
```

```

fragment@1 {
    target = <&ocp>;

    __overlay__ {
        test_helper: helper {

            compatible = "bone-pinmux-helper";

            pinctrl-names = "default";

            pinctrl-0 = <&mygpio>;

            status = "okay";

        };

    };
};

```

```

fragment@2{
    target = <&pruss>;

    __overlay__ {

        status = "okay";

    };

};

```

exclusive-use =

"P8.12";

=>

pin 12 on connector P8

pinctrl-single,pins = <

0x30 0x06

>;

=>

0x30 is the offset of pin address

0x06 is the value of pin

Convert this file into a .dtbo file using:

```
dtc -@ -O dtb -o BB-BONE-PRU-00A0.dtbo BB-BONE-PRU-00A0.dts
```

reboot

```
export PINS=/sys/kernel/debug/pinctrl/44e10800.pinmux/pins
```

```
export SLOTS=/sys/devices/bone_capemgr.8/slots
```

```
cd /lib/firmware
```

```
cat $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-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONE-LT-HDMI
```

```
echo BB-BONE-PRU > $SLOTS
```

```
cat $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-L Bone-Black-HDMI,00A0,Texas Instrument,BB-BONE-LT-HDMI
```

```
8: ff:P-O-L Override Board Name,00A0,Override Manuf,BB-BONE-PRU
```

To verify if the pins are configured correctly:

Type

```
cat $PINS | grep 830
```

then you will get

```
"pin 12 (44e10830) 00000006 pinctrl-single"
```

Or

```
cd /sys/kernel/debug/pinctrl/44e10800.pinmux
nano pins
```

In the table you will find this line:

```
pin 12 (44e10830) 00000006 pinctrl-single
                        address      value
```

Step 3: Assembler code and c code

A **General way** to make a new PRU application (*this method is based the existing example_apps in the zip file you've downloaded. Another method will be introduced in the PWM example.*)

in root@BBB:~/am335x_pru_package/pru_sw/example_apps# type:

```
cp -r PRU_memAccess_DDR_PRUsharedRAM mytest
```

go to the mytest directory, and change all " PRU_memAccess_DDR_PRUsharedRAM" to "mytest" by using *mv*

open mytest.p and make some changes:

```
" #include PRU_memAccess_DDR_PRUsharedRAM.hp" to " #include mytest.hp"
```

open mytest.c, and change all " PRU_memAccess_DDR_PRUsharedRAM" to "mytest"

open Makefile, and change all " PRU_memAccess_DDR_PRUsharedRAM" to "mytest"

go to am335x_pru_package/pru_sw/example_apps and nano DIRS: add mytest accordingly

```

192.168.1.104 - PuTTY
GNU nano 2.2.5 File: DIRS

APP_DIRS :=

APP_DIRS += PRU_memAccess_DDR_PRUsharedRAM
APP_DIRS += PRU_memAccessPRUDataRam
APP_DIRS += PRU_PRUtoPRU_Interrupt
APP_DIRS += mytest

ASSEM_FILES :=

ASSEM_FILES += PRU_memAccess_DDR_PRUsharedRAM/PRU_memAccess_DDR_PRUsharedRAM.p
ASSEM_FILES += PRU_memAccessPRUDataRam/PRU_memAccessPRUDataRam.p
ASSEM_FILES += PRU_PRUtoPRU_Interrupt/PRU_PRUtoPRU1_Interrupt.p
ASSEM_FILES += PRU_PRUtoPRU_Interrupt/PRU_PRUtoPRU0_Interrupt.p
ASSEM_FILES += mytest/mytest.p

BIN_FILES :=

BIN_FILES += PRU_memAccess_DDR_PRUsharedRAM.bin
BIN_FILES += PRU_memAccessPRUDataRam.bin
BIN_FILES += PRU_PRUtoPRU1_Interrupt.bin
BIN_FILES += PRU_PRUtoPRU0_Interrupt.bin
BIN_FILES += mytest.bin

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell

```

Change the mytest.c and mytest.p code to what you want (or you can download the source code from this website, see *Attachments*).

and then in am335x_pru_package/pru_sw/example_apps type:

make clean

make

and then copy the mytest.bin in directory am335x_pru_package/pru_sw/example_apps/bin to am335x_pru_package/pru_sw/example_apps/mytest/obj

and then in directory am335x_pru_package/pru_sw/example_apps/mytest/obj:

gcc mytest.o -L../././app_loader/lib -lprussdrv -lpthread -o mytest.out

execute

./mytest.out

then you will see

INFO: Starting mytest example.

AM33XX

INFO: Initializing example.

INFO: Executing example.

File ./mytest.bin open passed

INFO: Waiting for HALT command.

INFO: PRU completed transfer.

Example executed succesfully.

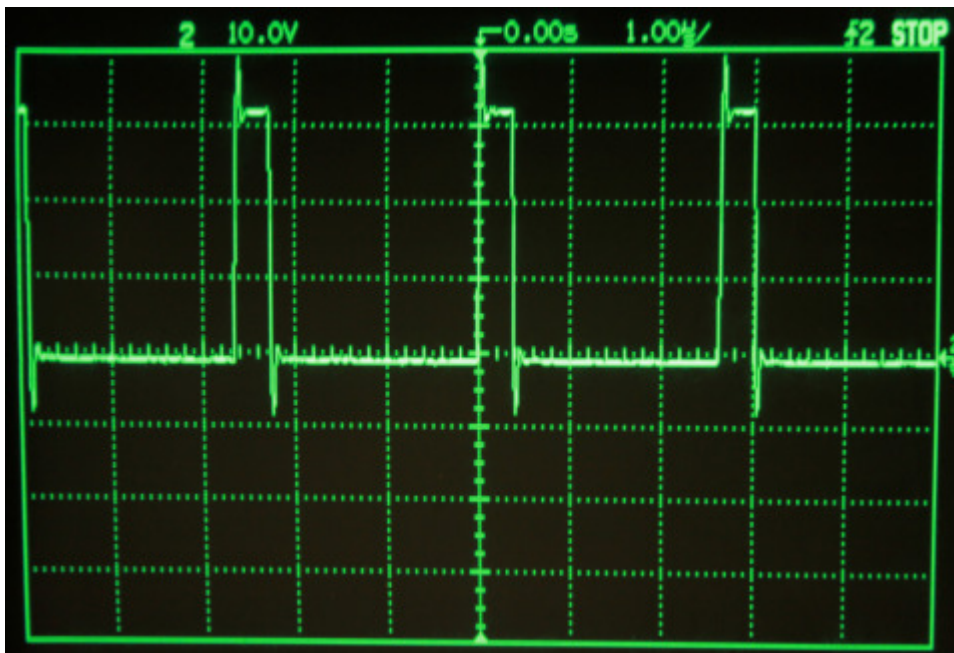
and the led will flash 10 times

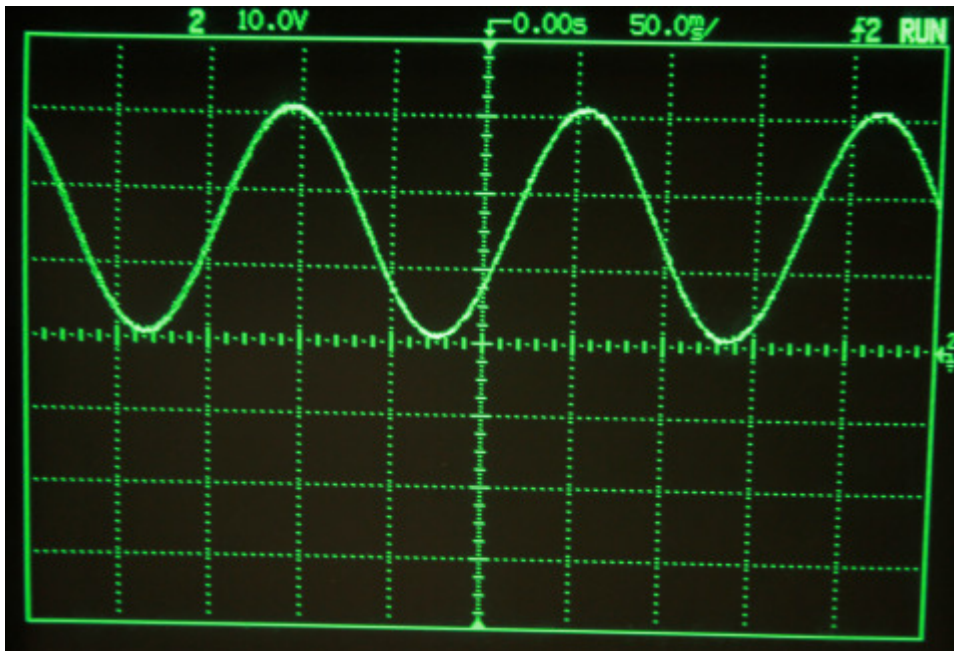
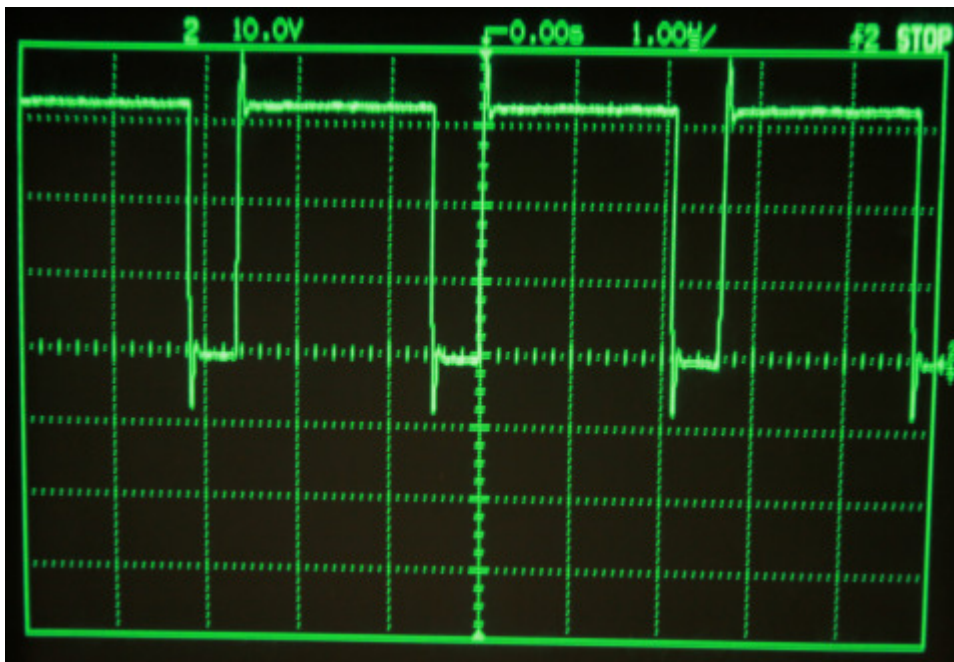
Attachment1: source code located in the folder 'led' in example_apps which can be downloaded from Example 3

[pru.hp .docx](#)

Example 2 - PRU0 PWM

This example use the same PRU and the same output pin as in example 1. We will skip all the details about how to configure the pin and the linux device tree. However, we will introduce another way(different from the way in step 3 of example 1) to compile the c code and assembler code.





About the sine wave, you need to add a **low-pass filter** to the digital pwm output pin. The low-pass filter can be as simple as a resistor-capacitor pair.

Another way to compile the c code and assembler code.

To make sure that you can compile the files in any folder, do the following:

copy the library and header files to system folders:

Go to pru_sw/app_loader/lib

```
cp libprussdrv.a /usr/lib/.
```

```
cd ../include
```

```
cp *.h /usr/include
```

```
cd pru_sw/utlis
```

cp pasm /usr/bin (or use pasm_2 if there's no pasm, since we've changed the pasm to pasm_2 in step 7 of 'Getting started'.)

compile the assembler code

```
pasm -V3 -cdl -b pwm.p
```

compile c code and generate executable file

```
gcc pwm.c -Wall -I ../../app_loader/include -D__DEBUG -O2 -mtune=cortex-a8 -march=armv7-a -o pwm -L  
../../app_loader/lib -lprussdrv -lpthread -lm
```

Note:

It is recommended to put the pwm folder in the same path of other examples making sure that the header files are included properly.

Attachment2: [pwm.zip](#)

Example 3 - Enable 8 output pins on PRU1

The configuration for **PRU1** is a little **different** from that for **PRU0**. Details are referred to mytest.c and mytest.hp

Configure the output pins for PRU1

In order to use the 8 pins of the LCD, you need to disable the HDMI Cape by adding the following marked text in uEnv.txt:

```
optargs=quiet capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN
```

The pins are listed below:

PINMUX

```
MUX_VAL(CONTROL_PADCONF_LCD_DATA0, (IDIS | OFF | MODE5 )) /* pr1_pru1_pru_r30[0] */  
MUX_VAL(CONTROL_PADCONF_LCD_DATA1, (IDIS | OFF | MODE5 )) /* pr1_pru1_pru_r30[1] */  
MUX_VAL(CONTROL_PADCONF_LCD_DATA2, (IDIS | OFF | MODE5 )) /* pr1_pru1_pru_r30[2] */  
MUX_VAL(CONTROL_PADCONF_LCD_DATA3, (IDIS | OFF | MODE5 )) /* pr1_pru1_pru_r30[3] */  
MUX_VAL(CONTROL_PADCONF_LCD_DATA4, (IDIS | OFF | MODE5 )) /* pr1_pru1_pru_r30[4] */  
MUX_VAL(CONTROL_PADCONF_LCD_DATA5, (IDIS | OFF | MODE5 )) /* pr1_pru1_pru_r30[5] */  
MUX_VAL(CONTROL_PADCONF_LCD_DATA6, (IDIS | OFF | MODE5 )) /* pr1_pru1_pru_r30[6] */  
MUX_VAL(CONTROL_PADCONF_LCD_DATA7, (IDIS | OFF | MODE5 )) /* pr1_pru1_pru_r30[7] */
```

MUX

```
#define CONTROL_PADCONF_LCD_DATA0    0x08A0  
#define CONTROL_PADCONF_LCD_DATA1    0x08A4
```

```
#define CONTROL_PADCONF_LCD_DATA2    0x08A8
#define CONTROL_PADCONF_LCD_DATA3    0x08AC
#define CONTROL_PADCONF_LCD_DATA4    0x08B0
#define CONTROL_PADCONF_LCD_DATA5    0x08B4
#define CONTROL_PADCONF_LCD_DATA6    0x08B8
#define CONTROL_PADCONF_LCD_DATA7    0x08BC
```

BB-BONE-PRU-00A0.dts are modified to

```
exclusive-use =
```

```
"P8.12","P8.45","P8.46","P8.43","P8.44","P8.41","P8.42","P8.39","P8.40";
```

and

```
pinctrl-single,pins = <
    0x30 0x06 0xa0 0x05 0xa4 0x05 0xa8 0x05 0xac 0x05 0xb0 0x05 0xb4 0x05 0xb8 0x05 0xbc 0x05
    >;
```

Connect leds to the configured output pins, then you will see blinking leds.

Attachment3: [example_apps.zip](#)

Example 4 - Using one single pointer to access all data RAMs

In this example we will show you how to access all data RAMS (data RAM 0, data RAM 1 and shared RAM) with one single pointer, which means that the cumbersome memory mapping is no more needed or can be dramatically simplified.

Local & Global Memory Map

Local Memory Map

- Allows PRU to directly access subsystem resources, e.g. DRAM, INTC registers, etc.
- NOTE: Memory map slightly different from PRU0 and PRU1 point-of-view.

Start	End	PRU/SS
0xA10 0000	0xA10 1FFF	Data 8KB RAM 0
0xA10 2000	0xA10 3FFF	Data 8KB RAM 1
0xA10 4000	0xA10 5FFF	Data 12KB RAM 2
0xA10 6000	0xA10 6FFF	INTC
0xA10 7000	0xA10 7FFF	PRU0 Control Registers
0xA10 8000	0xA10 8FFF	PRU0 Control Registers
0xA10 9000	0xA10 9FFF	PRU1 Control
0xA10 A000	0xA10 AFFF	PRU1 Control
0xA10 B000	0xA10 BFFF	PRU1 Control
0xA10 C000	0xA10 CFFF	PRU1 Control
0xA10 D000	0xA10 DFFF	PRU1 Control
0xA10 E000	0xA10 EFFF	PRU1 Control
0xA10 F000	0xA10 FFFF	PRU1 Control
0xA10 0000	0xA10 0FFF	PRU1 Control
0xA10 1000	0xA10 1FFF	PRU1 Control
0xA10 2000	0xA10 2FFF	PRU1 Control
0xA10 3000	0xA10 3FFF	PRU1 Control
0xA10 4000	0xA10 4FFF	PRU1 Control
0xA10 5000	0xA10 5FFF	PRU1 Control
0xA10 6000	0xA10 6FFF	PRU1 Control
0xA10 7000	0xA10 7FFF	PRU1 Control
0xA10 8000	0xA10 8FFF	PRU1 Control
0xA10 9000	0xA10 9FFF	PRU1 Control
0xA10 A000	0xA10 AFFF	PRU1 Control
0xA10 B000	0xA10 BFFF	PRU1 Control
0xA10 C000	0xA10 CFFF	PRU1 Control
0xA10 D000	0xA10 DFFF	PRU1 Control
0xA10 E000	0xA10 EFFF	PRU1 Control
0xA10 F000	0xA10 FFFF	PRU1 Control

Start	End	PRU0	PRU1
0x0000 0000	0x0000 1FFF	PRU0 Instruction RAM	PRU1 Instruction RAM

Start	End	PRU0	PRU1
0x0000 0000	0x0000 1FFF	Data 8KB RAM 0	Data 8KB RAM 1
0x0000 2000	0x0000 3FFF	Data 8KB RAM 1	Data 8KB RAM 0
0x0001 0000	0x0001 1FFF	Data 12KB RAM 2	Data 12KB RAM 2
0x0002 0000	0x0002 1FFF	INTC	INTC
0x0002 2000	0x0002 2FFF	PRU0 Control Registers	PRU0 Control Registers
0x0002 3000	0x0002 3FFF	Reserved	Reserved
0x0002 4000	0x0002 4FFF	PRU1 Control	PRU1 Control
0x0002 5000	0x0002 5FFF	Reserved	Reserved
0x0002 6000	0x0002 6FFF	CFG	CFG
0x0002 7000	0x0002 7FFF	UART 0	UART 0
0x0002 8000	0x0002 8FFF	Reserved	Reserved
0x0002 9000	0x0002 9FFF	Reserved	Reserved
0x0002 A000	0x0002 AFFF	Reserved	Reserved
0x0002 B000	0x0002 BFFF	Reserved	Reserved
0x0002 C000	0x0002 CFFF	Reserved	Reserved
0x0002 D000	0x0002 DFFF	Reserved	Reserved
0x0002 E000	0x0002 EFFF	Reserved	Reserved
0x0002 F000	0x0002 FFFF	Reserved	Reserved
0x0003 0000	0x0003 0FFF	Reserved	Reserved
0x0003 1000	0x0003 1FFF	Reserved	Reserved
0x0003 2000	0x0003 2FFF	Reserved	Reserved
0x0003 3000	0x0003 3FFF	Reserved	Reserved
0x0003 4000	0x0003 4FFF	Reserved	Reserved
0x0003 5000	0x0003 5FFF	Reserved	Reserved
0x0003 6000	0x0003 6FFF	Reserved	Reserved
0x0003 7000	0x0003 7FFF	Reserved	Reserved
0x0003 8000	0x0003 8FFF	Reserved	Reserved
0x0003 9000	0x0003 9FFF	Reserved	Reserved
0x0003 A000	0x0003 AFFF	Reserved	Reserved
0x0003 B000	0x0003 BFFF	Reserved	Reserved
0x0003 C000	0x0003 CFFF	Reserved	Reserved
0x0003 D000	0x0003 DFFF	Reserved	Reserved
0x0003 E000	0x0003 EFFF	Reserved	Reserved
0x0003 F000	0x0003 FFFF	Reserved	Reserved

Global Memory Map

- Allows external masters to access PRU subsystem resources, e.g. debug and control registers.
- PRU cores can also use global memory map, but more latency since access routed externally.

```
/******
```

```
* Local Macro Declarations *
```

```
*****/
```

```
#define PRU_NUM 1
```

```
#define OFFSET_MEM0 0x00002000
```

```
#define OFFSET_SHARED RAM 0x00010000
```

```
/******
```

```
* Local Function Declarations *
```

```
*****/
```

```
static int LOCAL_exampleInit();
```

```
/******
```

```
* Global Variable Definitions *
```

```
*****/
```

```
static void *pruDataMem;
```

```
static unsigned short *pruDataMem_short0; //AM33XX_DATA 8KB RAM0
```

```
static unsigned short *pruDataMem_short1; //AM33XX_DATA 8KB RAM1
```

```
prussdrv_map_prumem(PRUSS0_PRU1_DATARAM, &pruDataMem);
```

```
//assign the the data RAM address to two pointers
```

```
pruDataMem_short1 = (unsigned short*)pruDataMem; //AM33XX_DATA 8KB RAM1, Global Memory Address
```

```
pruDataMem_short0 = (unsigned short*)(pruDataMem - OFFSET_MEM0); //AM33XX_DATA 8KB
RAM0, Global Memory Address
```

Now with these two pointers you can access the two data RAMs. You can also change the pointer type to *'unsigned int'* or other types. You can also only use one pointer to access all the memories as long as you take the offset into account.

Attachment4: source code

Example 5 - 4 PWMs

BBB Schematic	BBB port	Assign	Bit
-----	-----	-----	-----
LCD_DATA0	P8.45	PWM0	PRU1_R30_0
LCD_DATA1	P8.46	PWM1	PRU1_R30_1
LCD_DATA2	P8.43	PWM2	PRU1_R30_2
LCD_DATA3	P8.44	PWM3	PRU1_R30_3

The pwms use the same pins of LCD or HDMI, so to avoid conflicting you need to disable the HDMI. Put the following code in your uEnv file:

```
optargs=quiet capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN
```

Change the Device Tree(DT):

In the zip file you can find a new BB-BONE-PRU-00A0.dts, copy this to /lib/firmware path.

Convert this file into a .dtbo file using:

```
dtc -@ -O dtb -o BB-BONE-PRU-00A0.dtbo BB-BONE-PRU-00A0.dts
```

reboot

Put the new pwm files in this path:

```
root@beaglebone:~/am335x_pru_package/pru_sw/example_apps/
```

Go to pwm folder and run:

```
pasm -V3 -cdl -b pwm.p
```



```
gcc pwm.c -Wall -I ../../app_loader/include -D__DEBUG -O2 -mtune=cortex-a8 -march=armv7-a -o pwm -L  
../../app_loader/lib -lprussdrv -lpthread -lm
```

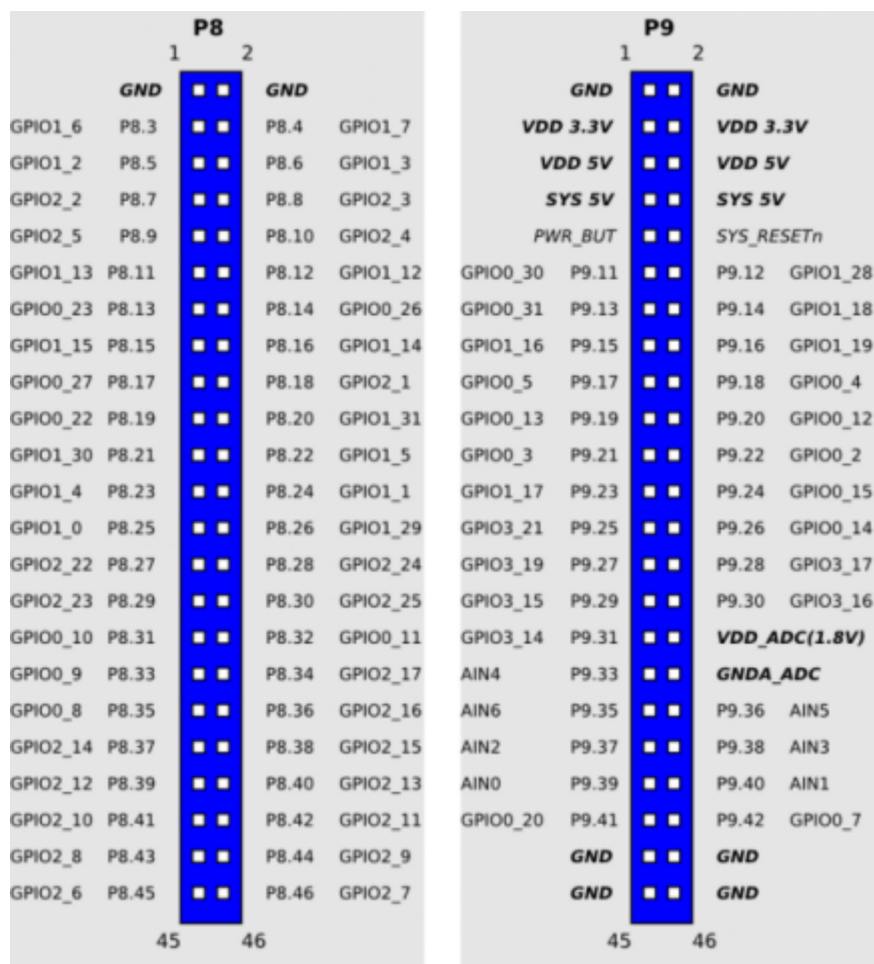
Or you can run the pre-built executable file: ./pwm

The PWMs' period T is 2.64us which corresponds to 8 bits. So the pulse width = (Duty cycle number)/256* T .

When running the code you will see:

```
root@beaglebone:~/am335x_pru_package/pru_sw/example_apps/pwm# ./pwm  
  
INFO: Starting PWM Demo with PRU example.  
AM33XX  
      INFO: Executing example.  
File ./pwm.bin open passed  
Choose PWM 0,1,2,3:  
0  
Delays:  
20000  
Duty cyle 0-256(0%-100%):  
124  
Pulse width(1.278750) us  
-> Start PWM yes/no (1/0)?  
1  
      INFO: Waiting for HALT command.  
      INFO: PRU completed transfer.
```

Attachment5: [pwms.zip](#)



Texas Instruments Wiki [PRU Linux Application Loader API Guide](#)

This article is inspired by shabaz with his article [BBB - Working with the PRU-ICSS/PRUSSv2](#).

FYI, if you are really in need of some help or have some suggestions for us, feel free to contact us welcome@analogdigitallab.org or comment on this page below.

Tags:

[BeagleBone Black](#)

[PRU-ICSS](#)

[PRU](#)

[device tree](#)

[BeagleBone](#)

[bbb](#)

[angstrom](#)

Comments

can you post more demo about pru?

Permalink On **Tue, 10/08/2013 - 21:00**

hi,

This is really a detailed write-up , great work.
Look forward to more applications of pru

By Geek

There will be more

Permalink On **Mon, 10/14/2013 - 20:10**

There will be more applications. we've achieved to interface pru to other device at 80Mbit/s. But the code and software still need to be optimized.

By Haolin Li

help with several AIs and PWMs

Permalink On **Sat, 12/07/2013 - 22:43**

hi,

first of all, congrats for your website and guidelines, they are really fantastic. I am not really familiarise with assembler. I have a beagle bone black, i followed all your examples and they work properly in my beagle bone.

As I said I am not familiarise with assembler and PRU. I wonder if anybody could help me on that. I need to use 4 pam and 4 AI in beagle bone, but I don't know exactly how to do it. I wonder if you could modificate pwm.c and pwm.p and pwm.hp to use this signals and also BB-BONE-PRU-00A0.dts to use this signal.

I understand that maybe you are quite busy with your own job. but I will really appreciate it if you can help me on that, because i am not sure if i should copy paste the same code several times or create a bucle to get the mod efficient way.

thank you so much and sorry for the inconveniences

regards

By **ignacio (not verified)**

AIs and PWMs

Permalink On **Tue, 12/10/2013 - 11:36**

hi **ignacio**,

can you specify more details on '4 pam and 4 AI in beagle bone' ? if you can tell us

more details, maybe we can directly translate it into assembler code.

By **Haolin Li**

Als and PWMs

Permalink On **Tue, 12/10/2013 - 15:31**

hi sorry

It was my fault. I wanted to say PWMs and analog inputs.

In your example we are using P8.12. I would like to have access to 4 PWM outputs, 4 analog inputs at least, if it is possible to add access to 4 digital inputs and 4 digital outputs will be fantastic. Basically I would like to use my BBB as embedded controller for my designs, but i don't know exactly how to develop the files *.p, *.hp, *.c and pru dts file BB-BONE-PRU-00A0.dts to acces to these signals.

if you can help me with this,i will really appreciate it, i guess it will take time and i am really sorry for disturbing you

Thanks

```

I1) BB-BONE-PRU-00A0.dts
/*
 * pru dts file BB-BONE-PRU-00A0.dts
 */
/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";

    /* identification */
    part-number = "BB-BONE-PRU";
    version = "00A0";
    exclusive-use =
        "P8.12";

    fragment@0 {
        target = &am33xx_pinmux;
        __overlay__ {
            mygpio: pinmux_mygpio{
                pinctrl-single,pins = <
                    0x30 0x06
                >;
            };
        };
    };

    fragment@1 {
        target = &ocp;
        __overlay__ {

```

```

    test_helper: helper {
        compatible = "bone-pinmux-helper";
        pinctrl-names = "default";
        pinctrl-0 = <&mygpio>;
        status = "okay";
    };
};
};

fragment@2{
    target = <&pruss>;
    __overlay__ {
        status = "okay";
    };
};
};

```

2) mytest.c

```

/*****
 * Include Files *
*****/

// Standard header files
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>
// #include <math.h>
// Driver header file
#include "prussdrv.h"
#include <pruss_intc_mapping.h>
#include <math.h>

/*****
 * Local Macro Declarations *
*****/

#define PRU_NUM 0

int main(void){

    unsigned int ret;
    unsigned int i,value;

    tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;
    printf("\nINFO: Starting %s example.\r\n", "PWM Demo with PRU");
    // Initialize the PRU
    prussdrv_init();
    // Open PRU Interrupt
    ret = prussdrv_open(PRU_EVTOUT_0);
    if (ret)

```

```

{
printf("prussdrv_open open failed\n");
return (ret);
}

// Get the interrupt initialized
prussdrv_pru_intc_init(&pruss_intc_initdata);

//Execute example on PRU
printf("\tINFO: Executing example.\r\n");
prussdrv_exec_program(PRU_NUM, "./mytest.bin");
    for (i = 0; i < 100000; i++) {
        usleep(100);
value=(sin(i*0.01)+1)*120+5;
prussdrv_pru_write_memory(PRUSS0_PRU0_DATARAM, 0x00000040, &value,
sizeof(unsigned int));
printf("(%d)\n-> ",value);
    }

    // Wait until PRU0 has finished execution
    printf("\tINFO: Waiting for HALT command.\r\n");
prussdrv_pru_wait_event(PRU_EVTOUT_0);
printf("\tINFO: PRU completed transfer.\r\n");
prussdrv_pru_clear_event(PRU0_ARM_INTERRUPT);

// Disable PRU and close memory mapping
prussdrv_pru_disable(PRU_NUM);
prussdrv_exit();
return(0);
}

```

3) mytest.p

```

.origin 0
.entriypoint START

#include "mytest.hp"

#define GPIO1 0x4804c000
#define GPIO_CLEARDATAOUT 0x190
#define GPIO_SETDATAOUT 0x194
#define MEM_START 0x00000100

START:
//set ARM such that PRU can write to GPIO
LBCO r0, C4, 4, 4
CLR r0, r0, 4
SBCO r0, C4, 4, 4

CLR r30,t14
SET r30,14

//for(r0=0x01000000;r0>0;ro--){
MOV r0,0x01000000

```



```

LOOP1:
  CLR r30.t14 //clear pin GPIO1_12

  MOV r18, MEM_START
  LBBO r1, r18, 0, 4
  AND r1, r1, 0x000000FF
DELAY2: SUB r1, r1, 1 //if r1=100 =>delay 1us
  QBNE DELAY2, r1, 0

  SET r30,14 //set pin GPIO44
  MOV r18, MEM_START
  LBBO r1, r18, 0, 4
  AND r1, r1, 0x000000FF
  MOV r2, 0x00000100
  RSB r1, r1, r2
DELAY3: SUB r1, r1, 1
  QBNE DELAY3, r1, 0

  SUB r0, r0, 1
  QBNE LOOP1, r0, 0

  MOV r31.b0, PRU0_ARM_INTERRUPT+16 //tell the c program we are
done
//(just remove it if your c program does not handle the interrupt)
  HALT // Halt the processor

```

By **Ignacio (not verified)**

PWMs

Permalink On **Sun, 12/15/2013 - 16:10**

hi Ignacio,

In example 5 you can find the source code of 4 PWMs which allows you to set duty cycles, delays and choose pwms. as for analog inputs i have no clue yet. If you want to access the analog inputs in PRU ,it will be a challenge. You can use the ARM processor itself instead of PRU. you can find some source code on the Internet to do this.

The other digital inputs and outputs are easy to implement if you follow the example 1.

By **Haolin Li**

-

Permalink On **Sun, 12/15/2013 - 22:24**

thank so much. I got an error when I wrote "echo BB-BONE-PRU > \$SLOTS"
echo:write error: File exists

I guess is due to one of the ports is in use, probably in conflict with HDMI?
Could you help me how to solve this?

i found this: For testing, try to cat the cape manager:

```
cat /sys/devices/bone_capemgr.*/slots  
find the number with the HDMI cape, and write:  
echo - /sys/devices/bone_capemgr.*/slots
```

But I am afraid about doing something wrong.

I would like to generate signals, continuous signals, so i am going to put a low pass filter in each
pwm. Could you modify the code .c and .p to assign values directly from pwm.c? :-S

Thank you so much and sorry for the inconveniences

The pwms use the same pins of LCD or HDMI, so to avoid conflicting you need
to disable the HDMI. put the following code in your uEnv file:

```
optargs=run_hardware_tests quiet  
optargs=quiet capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN
```

By **Haolin Li**

Very useful

Permalink On **Wed, 12/18/2013 - 15:32**

Great article, and lots of detailed examples that are really helpful. This is a great reference
that I can refer too. I am very glad you extended the original article! The PRUs are a very
useful resource inside the AM3359 chip so hopefully many people can use them with this
information.

By **Shabaz (not verified)**

Thank you and your inspiring

[Permalink](#) On **Thu, 12/19/2013 - 23:29**

Thank you and your inspiring article.

By [Haolin Li](#)

Using PRU for high performance audio

[Permalink](#) On **Wed, 12/18/2013 - 21:06**

I'm interested in using the Beaglebone Black as the DSP for a high fidelity digital preamp/room equalizer. I want to interface to an ADC (e.g. PM 1804) and a DAC (eg. PM1792a). These are 2 channel, 196kHz, 24 bit parts, which feature 3.3v I2S signalling. This requires about 10 Mbit/s in and out, which seems well within the PRU capabilities.

The one issue that worries me is that the system must never miss a sample. While the audio is streaming, the NEON FPU will be running a sophisticated digital filter, requiring on the order of 200 Mflops. The digital filter needs to fetch and store moderately large (~128 to 512 samples) from/to memory asynchronously from the PRUs.

Questions. Does this seem feasible? Can I directly connect the ADC and DAC to the GPIO pins (the logic voltages and protocols seem compatible)?

If this all works out, the result will be a really cheap but high quality preamp/equalizer. The ADC and DAC parts are under \$20 in small quantities.

By **John S. Rhoades (not verified)**

Using PRU for high performance audio

[Permalink](#) On **Mon, 12/23/2013 - 11:11**

Questions. Does this seem feasible? Can I directly connect the ADC and DAC to the GPIO pins (the logic voltages and protocols seem compatible)?

The output voltage of ADC is compatible with the PRU(3.3V). The output interface seems to be simple. The 10Mbit/s is achievable(we succeeded in interfacing a device up to 60Mit/s without counting the delay due to synchronization between PRU and C code, we need to transfer the data to ARM processor). I checked the datasheet, it seems the e.g. clock rising time is about 10ns, which won't be a problem for PRU running at 200MHz.

The PRU can definitely handle high data rate, what i concern is the synchroniztion between the PRU and c (I guess the digital filter uses c code to fetch and store data in PRU data RAM). You need to avoid the overwriting and some other issues that might occur in memory access.

One PRU can access all data RAMS (data RAM 0, data RAM 1 and shared RAM) (more referred to Example 4). So i assume the memory won't be a problem.

So,the most difficult part might be the data transfer between NEON FPU and PRU.

If it works out, i may want to buy such a cheap and high quality equalizer(just feel good to have good little devices around). :-)

Good luck

By **Haolin Li**

Using PRU for high performance audio

Permalink On **Mon, 12/30/2013 - 21:16**

The Beaglebone Black can perform a 4k x 4k convolution using the FFTW3 package out of the box in 20 ms. At 192 kHz, 4k samples equates to about 46 ms, which is a wide enough window for the filters I'm interested in. So, the Beaglebone has enough headroom to process 2 channels at 192 kHz in realtime, just barely. I've seen some notes on the web about getting a factor of 3 or 4 x improvement over this using custom NEON low level routines in FFTW3.

As far as I/O is concerned, I've been playing around with the PRU, and found it is loafing along with 8 channels in and out of 192 kHz audio. So it seems the project is feasible.

By **John S. Rhoades (not verified)**

Actually it turns out that

Permalink On **Wed, 01/08/2014 - 10:41**

Actually it turns out that the Beaglebone Black CPU+NEON can do a 4k x 4k convolution in 6.8 milliseconds! That's a measured result from my just-completed convolution code, not a theoretical prediction. The DFTs are handled by a modified FFTW3 package I found at <https://gitorious.org/gsoc2010-fftw-neon>, which comes from a Summer of Code project in 2010 to optimize FFTW3 for ARM/NEON processors. A 4096-tap FIR filter is more than adequate for the kinds of digital filters I want to implement. This represents a throughput of 301k stereo audio samples per second (2 channels x 4096 samples / 6.8 ms). Thus at 192 kHz, the convolution operation would consume about 2/3 of the available compute power of a Beaglebone Black. Since the convolution operation takes about 95% of the CPU cycles for the entire digital filter, there's enough compute power there to run in real time.

By **John S. Rhoades (not verified)**

PRU input problem

Permalink On **Sun, 12/22/2013 - 06:41**

Hi~ I ran into some small troubles when I was trying to get access to GPIO2 pins as inputs from PRU. My program just kept reading the GPIO_DATAIN register of a GPIO and sending them to PRU DRAM. It worked perfectly on GPIO1, but I got constant ZERO output when I tried to read any register of GPIO2 (as well as GPIO0 and GPIO3). Any ideas? Looking forward to your response. :)

By **wyt (not verified)**

PRU input problem

Permalink On **Mon, 12/23/2013 - 10:32**

hi

I used PRU1, 8 data input pins from GPIO2 (connector P8). The register is r31. I can correctly write large amount of data to PRU DataRam. I didn't try other GPIOs.

I suggest you to use the tool for configuration: PinMuxUtility_02_05_02_00.zip available from this [link](#). Then you can find the correct GPIOs and corresponding registers.

This tool has already been explained in Example 1.

By Haolin Li

Analig input pru

Permalink On **Sun, 12/22/2013 - 10:56**

Hi, :-) happy Christmas

I need some help. I got to modify the last example to work with 4 pwm at the same time. But i would like to read also 1 or 2 analog inputs thsthrough the same way. Any of you know how to modify the last exmple to add 1 or 2 analog input?

Thank you so much and sorry for the inconveniences

:-)

By **Ignacio (not verified)**

Analig input pru

Permalink On **Mon, 12/23/2013 - 10:43**

Merry Christmas.

I didn't find a way to configure or access the analog inputs via PRU directly. Even in the tool PinMuxUtility_02_05_02_00.zip, i only found the configuration for the GPIOs, none for the Analog Inputs.

But i found two links that might be helpful:

http://www.phys-x.org/rbots/index.php?option=com_content&view=article&id=105:lesson-2-beaglebone-black-read-analog-&catid=46:beaglebone-black&Itemid=81

<http://beagleboard-gsoc13.blogspot.be/2013/07/sampling-analogue-signals-using-adc-on.html>

There might be other examples using Python or C. If you need to feed the analog input data to PRU, you can try to read the analog inputs in Python or C, and then load the data to the PRU DataRAM in C, so that the PRU can do something with the data.

Another way is to use external ADCs and interface the ADC to PRU. The PRU can definitely handle this.

Good luck

By **Haolin Li**

Great article

Permalink On **Sun, 03/09/2014 - 12:16**

many thanks, this article helps me a lot.

By **Jerry (not verified)**

Hi,

Permalink On **Mon, 03/10/2014 - 09:59**

Hi,

I am new with Linux OS and currently studying for PRUSS on AM335x.
I got an error as below when performing step4.

```
$ export CROSS_COMPILE=  
$ make  
gcc -l. -Wall -I../include -c -O3 -mtune=cortex-a8 -march=armv7-a -o release/prussdrv.o  
prussdrv.c  
prussdrv.c:1:0: error: bad value (armv7-a) for -march= switch  
prussdrv.c:1:0: error: bad value (cortex-a8) for -mtune= switch  
make: *** [release/prussdrv.o] Error 1
```

I have no idea what error is it.
Anyone can help me on this?
Thanks in advanced.

By **susan (not verified)**

hi Susan,

Permalink On **Mon, 03/10/2014 - 15:33**

hi Susan,

- 1) which Linux are you using? are you using beaglebone or beaglebone black?
- 2) if you are using Angstrom, kernel 3.8 on beaglebone black, normally it won't cause a

problem.

Try once the steps without update or upgrade your beaglebone black. Sometimes updating or upgrading will cause problems.

3)

Did you go back to path "go to pru_sw/app_loader/interface" after "\$ export CROSS_COMPILE=" ?

Let me know if these tips don't work.

Joris

By [joris van kerre...](#)

Hey

Permalink On **Mon, 03/24/2014 - 16:40**

First off thanks a lot for the detailed post. I was following the steps for setting it all up when i keep getting a problem at step 14: where I have to execute the mytest.out file. I keep getting this error right after typing ./mytest.out

./mytest.out: error while loading shared libraries: libprussdrv.so: cannot open shared object file: No such file or directory

To the best of my knowledge I followed all of the instructions above that. I'm working on ubuntu and quite new to linux itself. Hope I can get this sorted out.

Thanks!
Ayoub

By **Ayoub (not verified)**

libprussdrv.so

Permalink On **Mon, 03/24/2014 - 18:56**

I've never used pru on Ubuntu, only **Angstrom**. This article is only for the PRU on the Angstrom, i'm not sure if the Ubuntu can support the PRU.

Under step 14, there is a 'Solution' you can try this solution out. Or you can go to Example 1 and follow the instruction there. In the Example 1 you need to make a device tree, like **BB-BONE-PRU-00A0.dts**.

Please let me know if it is still not working.

By [Haolin Li](#)

Sorry I had other things to

Permalink On **Sun, 04/13/2014 - 07:30**

Sorry I had other things to take care of so couldnt get back to trying this method up until now. I managed to fix the libprussdrv.so error by just copying the library files from app_loader/lib to lib/ in root. So now everything works fine, even on the beaglebone white.

Thanks again!

By **Ayoub (not verified)**

Pin modes after enabling overlay

Permalink On **Tue, 06/03/2014 - 11:56**

First of all, I would like to thank you for a great job on showing some good examples on using the PRU.

I have a question about the mode settings. In the .dts file the pin modes are

exclusive-use =

"P8.12","P8.45","P8.46","P8.43","P8.44","P8.41","P8.42","P8.39","P8.40";

fragment@0 {

target = <am33xx_pinmux>;

__overlay__ {

mygpio: pinmux_mygpio{

pinctrl-single,pins = <

0x30 0x06 0xa0 0x05 0xa4 0x05 0xa8 0x05 0xaC 0x05 0xb0 0x05 0xb4 0x05 0xb8 0x05 0xbC 0x05

>;

After enabling the overlay, here 's what I see

root@beaglebone:~/pru/analog_digital_labs/pwm/example3# cat \$PINS |grep 830

pin 12 (44e10830) 00000006 pinctrl-single

root@beaglebone:~/pru/analog_digital_labs/pwm/example3# cat \$PINS |grep 8a0

pin 40 (44e108a0) 0000002f pinctrl-single

root@beaglebone:~/pru/analog_digital_labs/pwm/example3# cat \$PINS |grep 8a4

pin 41 (44e108a4) 0000002f pinctrl-single

If I am not mistaken pin 40 is P8_45 (offset of a0) and should have a mode of 05 but why is it showing 2f ? The same thing with pin 41?

By **cyrus (not verified)**

hi cyrus,

Permalink On **Tue, 06/03/2014 - 13:10**

hi cyrus,

did you disable the HDMI? 0x2f means GPIO mode with no pullup/pulldown.

By **Haolin Li**

Yes, I did. I disabled it at

Permalink On **Wed, 06/04/2014 - 14:09**

Yes, I did. I disabled it at startup.

By **cyrus (not verified)**

/boot/am335x-boneblack.dtb file does not exist!

Permalink On **Sat, 09/13/2014 - 10:02**

Hello Sir. Thanks for this tutorial. I was following the steps and I reached step-14. I got the error which you said I will need some modifications. I changed the current directory to /boot but there was no file named "am335x-boneblack.dtb". Any idea how to get this to work? I am using this version of linux: Linux beaglebone 3.8.13-bone50 #1 SMP Tue May 13 13:24:52 UTC 2014 armv7l GNU/Linux

Another question, I want to read from all 7 analog inputs at a very high sampling rate. I wrote several codes in python, javascript, and using terminal but I could not reach a sampling rate more than 1KHz!

Thanks in advance.

By **Mahmoud (not verified)**

/boot/am335x-boneblack.dtb file does not exist!

Permalink On **Sat, 09/13/2014 - 10:02**

Hello Sir. Thanks for this tutorial. I was following the steps and I reached step-14. I got the error which you said I will need some modifications. I changed the current directory to /boot but there was no file named "am335x-boneblack.dtb". Any idea how to get this to work? I am using this version of linux: Linux beaglebone 3.8.13-bone50 #1 SMP Tue May 13 13:24:52 UTC 2014 armv7l GNU/Linux

Another question, I want to read from all 7 analog inputs at a very high sampling rate. I wrote several codes in python, javascript, and using terminal but I could not reach a sampling rate more than 1KHz!

Thanks in advance.

By **Mahmoud (not verified)**

it's in the /boot/dtbs/

Permalink On **Tue, 09/30/2014 - 11:05**

it's in the /boot/dtbs/
or you can use command tool: locate

By **longqi (not verified)**

process interrupt

Permalink On **Tue, 09/30/2014 - 11:03**

I notice there is a function in prussdrv, called "prussdrv_pru_wait_event", So that means when the pru running, Our C program which running on arm core can not do any other thing?

For the interrupt mapping, what's the "host" means? Do they refer to the pru0 pru1 and arm core? but why it have ten host number? I must missed something. I'm confused now.

By longqi (not verified)

device tree as startup

Permalink On **Tue, 10/28/2014 - 21:28**

hi

I would like to leave a device tree as startup, because everytime that i turn on my BB i have to follow the sequence:

```
export PINS=/sys/kernel/debug/pinctrl/44e10800.pinmux/pins
export SLOTS=/sys/devices/bone_capemgr.8/slots
cd /lib/firmware
cat $SLOTS
echo BB-BONE-PRU > $SLOTS
cat $SLOTS
```

any of you knows how to do that?

Thank you so much for your support

regards

By ignacio (not verified)

Is it possible modifying uEnv

Permalink On **Sun, 11/02/2014 - 14:52**

Is it possible modifying uEnv.txt? I was looking for some info in google but i could not find the solution.

Could you help me with this?

Thank you so much

By ignacio (not verified)

you could add that to the ~/

Permalink On **Tue, 05/26/2015 - 15:33**

you could add that to the ~/.profile file, it will then get executed at logon

By Sadat (not verified)

Can't open source file 'mytest.p'

[Permalink](#) On **Fri, 03/20/2015 - 17:17**

PRU Assembler Version 0.86

Copyright (C) 2005-2013 by Texas Instruments Inc.

Pass 2 : 0 Error(s), 0 Warning(s)

Writing Code Image of 9 word(s)

Hello,

Thank you so much for your support.

At the last of the example 1, when I make example_apps I get this error:

PRU Assembler Version 0.86

Copyright (C) 2005-2013 by Texas Instruments Inc.

Fatal Error: Can't open source file 'mytest.p'

Any idea about this problem plz?

By **MARS (not verified)**

ADC communication

[Permalink](#) On **Thu, 04/09/2015 - 16:13**

Hi, thank you for a detailed article.

I'm new in everything (BBB, linux and assembly), sadly.

But I have to do a project using the BBB.

First, I have to use a ADC to capture 8 signals simultaneously and continually. I want to use the PRU to communicate with the ADC and save the data in the shared memory continually. In paralel I'm gonna use the pricipal core of the BBB to process the data that it's comming.

The ADC is ADS8528 (12 bits) from texas instruments:

Maximum Data Rate Per Channel:

650kSPS (parallel) or 480kSPS (serial)

My questions are:

Which one is simpler to implement parallel ou serial with the PRU? And do you have a example simillar to I learn how to do that?

Many thanks.

By **Eddy (not verified)**

Device tree set, but the LED wont blink

[Permalink](#) On **Tue, 05/26/2015 - 15:32**

Hello,

Thanks a lot for the detailed write up, I am trying to use the PRU to controll GPIO, but I cant seem to get this thing working, I can use the PRU to toggle the GPIOs in more 7 but when I use the P8_12 in mode 6, it defaults to on! As in as soon as I echo the dtbo file, the LED is on, and then executing the assembler code as no change to the LED, I even have a oscilloscope, and that too shows nothing, could you please tell me what I could be doing wrong? I am useing the latest Linux kernal.

Thanks
Sadat

By **Sadat (not verified)**

Hello there!

Permalink On **Sun, 05/31/2015 - 19:19**

Hello there!

I am using HC-SR04 (Ultrasonic Sensors) with my beaglebone black I found out some codes and they work perfectly fine at P8_11 and P8_12.

I need to use 3 sensors simultaneously but I can't figure out where to make changes in .p file! I have figured out the .dts file.

the link is below:

<https://github.com/luigif/hcsr04/blob/master/README.md>

I have a project due and need assistance urgently please guide!

Thank-You!

By **Usama Zafar (not verified)**

Long real-time sequence with PRU

Permalink On **Fri, 07/17/2015 - 23:08**

I would be interested in the possibility of running a long sequence of hard-real-time events (output on GPIO).

The list would be too long to reside in the PRU memory but could be buffered by it (circular buffer?).

The main processor would have the whole list and feed it to the PRU as it runs through the list.

The main processor would need to be able to synchronize other things (send SPI data for example) with the sequence.

The whole sequence could run for a few days.

By **Cat (not verified)**