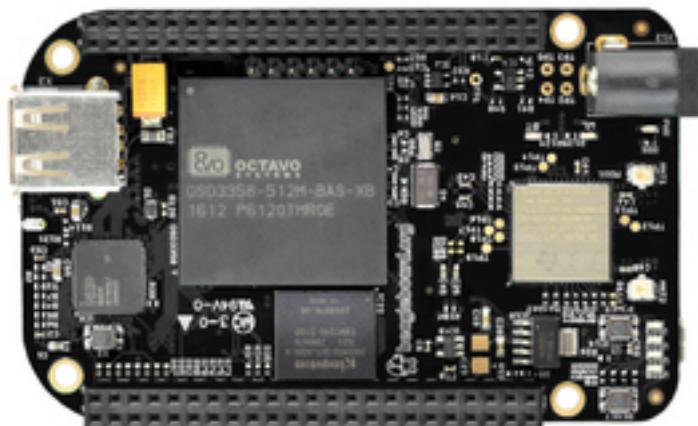


Using the PRUs and RPMsg

BEAGLEBONE™ BLACK WIRELESS LINUX DEBIAN 4.9.45-TI-R57



Pierrick RAUBY
Master Thesis Student

Contents

1	Introduction	1
2	Hardware presentation	2
3	Enabling the PRUs	4
3.1	Setting up the PRUs	4
3.1.1	Disabling the HDMI cape and loading the PRU overlay	4
3.1.2	Installing GCC compiler	5
3.1.3	Creating the symbolic links between folders	5
3.2	Testing the PRUs	6
3.2.1	Hardware	6
3.2.2	Code	7
3.2.3	Running the example	8
4	RPMsg	10
4.1	Presentation of RPMsg	10
4.2	Setup	11
4.3	Testing	11
4.3.1	Code for the Cortex-A8	12
4.3.2	Code for the PRU	13
4.3.3	Starting the project	14
A	PIN Header 8	16
B	PIN Header 9	17

Chapter 1

Introduction

The BeagleBone™ Black is a low-cost development platform powered by an AM335x 1GHz ARM® Cortex-A8, among its different features, the AM335x presents two Process Real Time Units (PRU). For my master thesis I will need to use those two micro-controllers in order to acquire data from an accelerometer, it took some time to enable the PRU and the communication framework: RPMsg. The purpose of this document, is to explain the method followed to enable those embedded micro-controllers and the framework.

Zubeen Tolani, Gregory Raven are acknowledged for the very complete documentation they have provided about the PRUs and the RPMsg framework which can be found here:

- [BeagleScope repository on GitHub from Zubeen Tolani](#)
- [PRU ADC repository on GitHub from Gregory Raven](#)

Chapter 2

Hardware presentation

For this project the board used is a *BeagleBone™ Black wireless* powered by an *Octavo Systems OSD3358* which characteristics are :

- 512 MB DDR3 RAM
- 4GB 8-bit eMMC on board flash storage
- 3D graphic accelerator
- Neon floating-point accelerator
- 2 PRU : 32-bit microcontrollers

The software used is the Debian image: *Linux Beaglebone™ 4.9.45-ti-r57*

As explained in the introduction the idea of the project is to use the PRUs to acquire data from a sensor and send them to the ARM® Cortex of the BeagleBone™. But what are the PRUs and the ARM®? Basically, the Octavo contains the TI AM335X chip which itself contains:

- 1 ARM® Cortex®-A8: This is the part of the chip that runs the Linux operating system. This microprocessor as a "computer" processor is not able to carry out real-time operations.
- 2 Process Real-time Units (PRU) that are microcontrollers such Arduino/Teensy ones. It means that they are able to execute real-time process, then the *programmable nature of the PRU, along with its access to pins, events and all system on chip (SoC) resources, provides flexibility in implementing fast real-time responses, specialized data handling operations* [TexasInstruments, 2017]. Thus, PRU are very useful when it comes to carry out time critical operations such as fast data acquisitions.

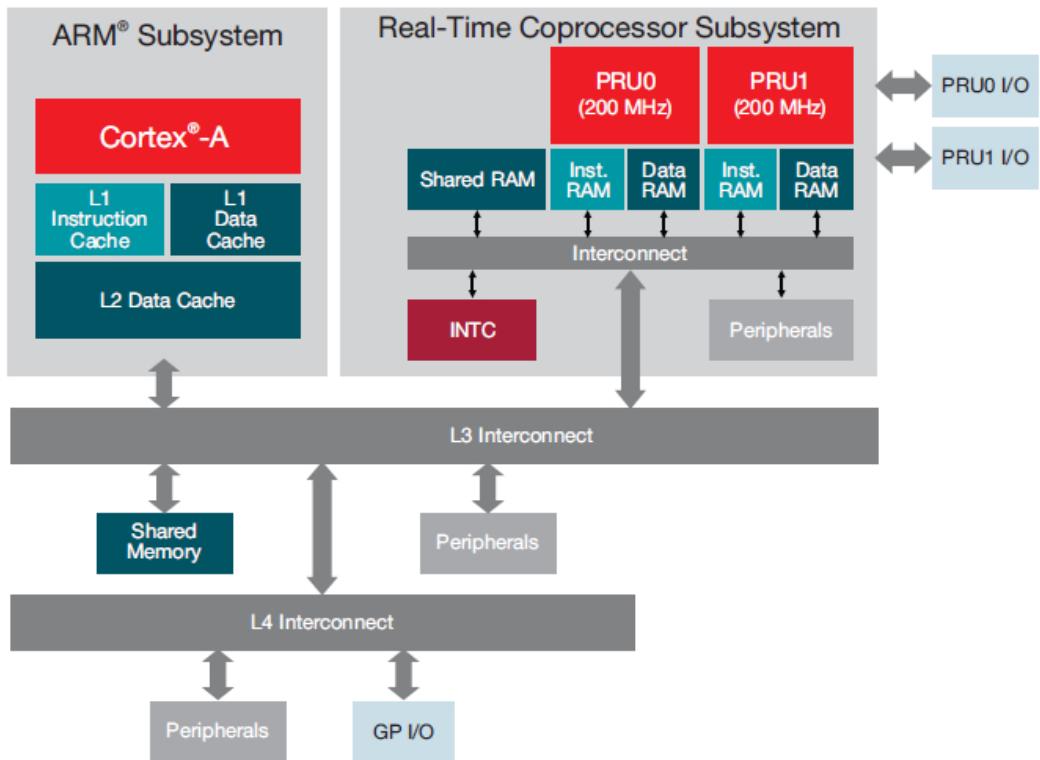


Figure 2.1: Architecture of the AM335x with Cortex®-A8 and the 2 PRUs [TexasInstrument, 2017a]

Chapter 3

Enabling the PRUs

In this chapter, the setup of the PRU is explained. The different steps are based on the work of [Tolani, 2016], who presents a very complete set of instruction in order to setup the PRUs for *Debian 4.4.12-ti-r31*, basically his work is adapted here for *Debian 4.9.45-ti-r57*.

3.1 Setting up the PRUs

3.1.1 Disabling the HDMI cape and loading the PRU overlay

The PRUs have access to many pins on the BeagleBone™, however some pins are also used by the HDMI. Thus, the HDMI must be disable before using the PRUs [Yoder, 2017]. In order to do so we are going to disable the loading of the device tree corresponding to the HDMI.

Remark: *The Device Tree (DT), and Device Tree Overlay are a way to describe hardware in a system. An example of this would be to describe how the UART or HDMI interfaces with the system, which pins, how they should be mixed, the device to enable, and which driver to use* [Cooper, 2015].

First of all, you need to SSH into the BeagleBone™ Black as root, then navigate to the uEnv.txt file by typing in:

```
cd /boot/  
nano uEnv.txt
```

Then the uEnv.txt file should appear as in figure 3.1:

```
#Docs: http://elinux.org/Beagleboard:U-boot_partitioning_layout_2.0  
  
uname_r=4.9.45-ti-r57  
#uuid=  
#dtb=  
  
###U-Boot Overlays###  
###Documentation: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian#U-Boot_Overlays  
###Master Enable
```

Figure 3.1: uEnv.txt

In this file, you should go down to the section,

```
###Disable auto loading of virtual capes (emmc/video/wireless/adc)
```

and uncomment the two lines as shown below, this avoids the loading of the HDMI overlays at boot time:

```
###Disable auto loading of virtual capes (emmc/video/wireless/adc)
#disable_uboot_overlay_emmc=1
disable_uboot_overlay_video=1
disable_uboot_overlay_audio=1
#disable_uboot_overlay_wireless=1
#disable_uboot_overlay_adc=1
###
```

In the same document we are going to ask for the loading of the PRUSS overlay at boot time, scroll down to the section:

```
###PRUSS OPTIONS
###pru_rproc (4.4.x-ti kernel)
```

change these lines:

```
###PRUSS OPTIONS
###pru_rproc (4.4.x-ti kernel)
#uboot_overlay_pru=/lib/firmware/AM335X-PRU-RPROC-4-4-TI-00A0.dtbo
###pru_uio (4.4.x-ti & mainline/bone kernel)
#uboot_overlay_pru=/lib/firmware/AM335X-PRU-UIO-00A0.dtbo
###
```

to:

```
###PRUSS OPTIONS
###pru_rproc (4.4.x-ti kernel)
uboot_overlay_pru=/lib/firmware/AM335X-PRU-RPROC-4-9-TI-00A0.dtbo
###pru_uio (4.4.x-ti & mainline/bone kernel)
#uboot_overlay_pru=/lib/firmware/AM335X-PRU-UIO-00A0.dtbo
###
```

3 modifications: 1 uncomment, 1 comment and the 4-4-TI-00A0.dtbo becomes 4-9-TI-00A0.dtbo

Finally, just reboot the board. The HDMI capes should be disabled so we have access to the different PINs of the board with the PRU, figure 3.2 presents the PIN for the Header 8 (more details on appendix A and B).

3.1.2 Installing GCC compiler

Since the PRUs are based on TI's proprietary architecture [Tolani, 2016], we have to compile the C code that we want to execute with a compiler. In this project GCC is used.

```
cd
wget -c http://software-dl.ti.com/codegen/esd/cgt_public_sw/PRU/2.1.2/
      ti_cgt_pru_2.1.2_armlinuxa8hf_busybox_installer.sh
chmod +x ti_cgt_pru_2.1.2_armlinuxa8hf_busybox_installer.sh
./ti_cgt_pru_2.1.2_armlinuxa8hf_busybox_installer.sh
cd
rm ti_cgt_pru_2.1.2_armlinuxa8hf_busybox_installer.sh
```

3.1.3 Creating the symbolic links between folders

Then, some symbolic links have to be create:

```
cd /usr/share/ti/cgt-pru/
mkdir bin
cd
ln -s /usr/bin/clpru /usr/share/ti/cgt-pru/bin/clpru
ln -s /usr/bin/lnkpru /usr/share/ti/cgt-pru/bin/lnkpru
```

P8_20	33	0x884/084	63	GPIO1_31	gpio1[31]	pr1_pru1_pru_r31_13	pr1_pru1_pru_r30_13
P8_21	32	0x880/080	62	GPIO1_30	gpio1[30]	pr1_pru1_pru_r31_12	pr1_pru1_pru_r30_12
P8_22	5	0x814/014	37	GPIO1_5	gpio1[5]		
P8_23	4	0x810/010	36	GPIO1_4	gpio1[4]		
P8_24	1	0x804/004	33	GPIO1_1	gpio1[1]		
P8_25	0	0x800/000	32	GPIO1_0	gpio1[0]		
P8_26	31	0x87c/07c	61	GPIO1_29	gpio1[29]		
P8_27	56	0x8e0/0e0	86	GPIO2_22	gpio2[22]	pr1_pru1_pru_r31_8	pr1_pru1_pru_r30_8
P8_28	58	0x8e8/0e8	88	GPIO2_24	gpio2[24]	pr1_pru1_pru_r31_10	pr1_pru1_pru_r30_10
P8_29	57	0x8e4/0e4	87	GPIO2_23	gpio2[23]	pr1_pru1_pru_r31_9	pr1_pru1_pru_r30_9
P8_30	59	0x8ec/0ec	89	GPIO2_25	gpio2[25]		
P8_31	54	0x8d8/0d8	10	UART5_CTSN	gpio0[10]	uart5_ctsn	
P8_32	55	0x8dc/0dc	11	UART5_RTSN	gpio0[11]	uart5_rtsn	
P8_33	53	0x8d4/0d4	9	UART4_RTSN	gpio0[9]	uart4_rtsn	
P8_34	51	0x8cc/0cc	81	UART3_RTSN	gpio2[17]	uart3_rtsn	
P8_35	52	0x8d0/0d0	8	UART4_CTSN	gpio0[8]	uart4_ctsn	
P8_36	50	0x8c8/0c8	80	UART3_CTSN	gpio2[16]	uart3_ctsn	
P8_37	48	0x8c0/0c0	78	UART5_RXD	gpio2[14]	uart2_ctsn	
P8_38	49	0x8c4/0c4	79	UART5_RXD	gpio2[15]	uart2_rtsn	
P8_39	46	0x8b8/0b8	76	GPIO2_12	gpio2[12]	pr1_pru1_pru_r31_6	pr1_pru1_pru_r30_6
P8_40	47	0x8bc/0bc	77	GPIO2_13	gpio2[13]	pr1_pru1_pru_r31_7	pr1_pru1_pru_r30_7
P8_41	44	0x8b0/0b0	74	GPIO2_10	gpio2[10]	pr1_pru1_pru_r31_4	pr1_pru1_pru_r30_4
P8_42	45	0x8b4/0b4	75	GPIO2_11	gpio2[11]	pr1_pru1_pru_r31_5	pr1_pru1_pru_r30_5
P8_43	42	0x8a8/0a8	72	GPIO2_8	gpio2[8]	pr1_pru1_pru_r31_2	pr1_pru1_pru_r30_2
P8_44	43	0x8ac/0ac	73	GPIO2_9	gpio2[9]	pr1_pru1_pru_r31_3	pr1_pru1_pru_r30_3
P8_45	40	0x8a0/0a0	70	GPIO2_6	gpio2[6]	pr1_pru1_pru_r31_0	pr1_pru1_pru_r30_0
P8_46	41	0x8a4/0a4	71	GPIO2_7	gpio2[7]	pr1_pru1_pru_r31_1	pr1_pru1_pru_r30_1

Figure 3.2: P8 header and corresponding PRU [Molloy, 2014]

Finally, we want to that "PRU_CGT" point to the "/usr/share/ti/cgt-pru/":

```
export PRU_CGT=/usr/share/ti/cgt-pru
```

Because we want this last command to be executed every time we boot the Beaglebone™:

```
cd  
nano ~/.bashrc
```

and add this:

```
export PRU_CGT=/usr/share/ti/cgt-pru
```

Then save and quit and reboot.

3.2 Testing the PRUs

Now that everything is ready we can test the PRU with an "hello world!" example in which a small LED is trigger with the PRU. Let's create a small circuit with the LED and two resistors and copy the code testing codes on the BeagleBone™.

3.2.1 Hardware

The circuit used to test the PRU is presented on figure 3.3. Pin P8_45 is used as the output pin and pin P8_1 is connected to the ground of the circuit.

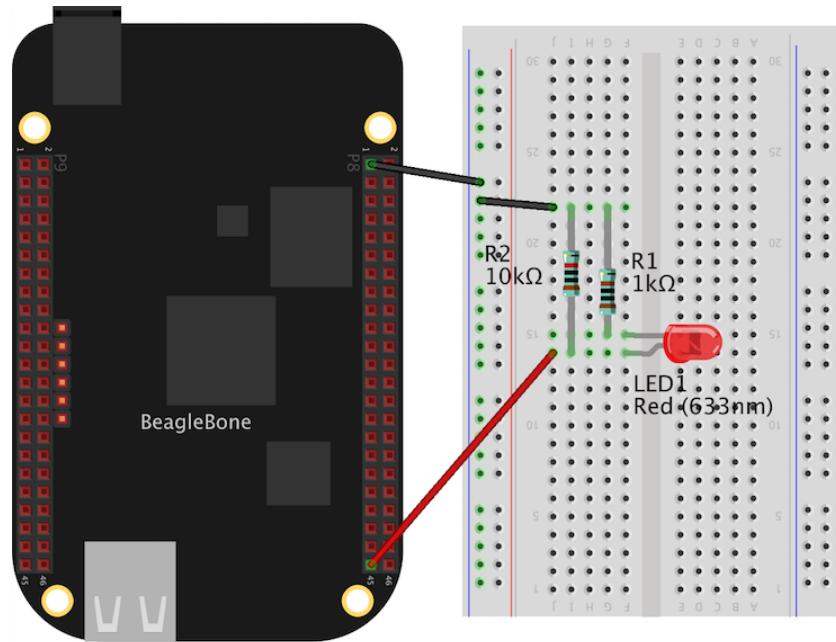


Figure 3.3: The circuit for Hello_PRU program

3.2.2 Code

Now that the hardware is ready, let's copy the code. First of all, go back to the "/root" folder of the BeagleBone™:

```
cd
```

And create a new folder "Hello_PRU":

```
mkdir Hello_PRU
```

In this folder we are going to put 5 files and 2 folders:

- Hello_PRU.c
- AM335x_PRU.cmd
- resource_table_empty.h
- Makefile
- deploy.sh
- lib which contains some needed libraries
- include which contains resources files for the different TI processors

3.2.2.1 Hello_PRU.c

This is the C code that once compiled with GCC is going to make our LED blink.

Lines 38 to 40 correspond to the inclusion of needed files. Lines 42 and 43 correspond to the declaration of two important registers, _R30 and _R31.

In the main loop (from line 45 to the end), the volatile "gpio" is used to toggle the value of the _R30 between 0x000F and 0x0000 waiting between each toggling thanks to the "_delay_cycles()" function (which is an intrinsic compiler function [Tolani, 2016]).

Remark: The compiler would not allow any variable other than _R31 and _R30 to be of the "register" type, and the compiler does not allow access to any of the 29-R0 registers of the PRU [Tolani, 2016].

```

38 #include <stdint.h>
39 #include <pru_cfg.h>
40 #include "resource_table_empty.h"
41
42 volatile register uint32_t __R30;
43 volatile register uint32_t __R31;
44
45 void main(void)
46 {
47     volatile uint32_t gpio;
48
49     /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
50     CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;
51
52     /* Toggle GPO pins TODO: Figure out which to use */
53     gpio = 0x000F;
54
55     /* TODO: Create stop condition, else it will toggle indefinitely */
56     while (1) {
57         __R30 ^= gpio;
58         __delay_cycles(100000000);
59     }
60 }
```

Figure 3.4: Hello_PRU.c code [Tolani, 2016]

3.2.2.2 AM335x_PRU.cmd

PRUs are pretty simple processing cores, but the PRUSS system is highly integrated and provides the PRU a rich set of peripherals. All these peripherals inside the PRUSS are at different address locations and they need to be configured by the Linux kernel at the time of firmware loading onto the PRUs. The "AM335x_PRU.cmd" file provides a mapping to the linker, from different sections of code, to different memory locations inside the PRUSS. [Tolani, 2016] Thus this file is a linker command file that is used for linking PRU programs built with the C compiler and the resulting .out file on an AM335x device. Basically, you will need this file every time you create a PRU code as the one above and compile it with GCC.

3.2.2.3 resource_table_empty.h

This empty resource table is needed by the "AM335x_PRU.cmd", it is used by Remoteproc, on the host-side to allocate reserved/resources. Since we do not use Remoteproc for the moment (but we will later) we just give an empty file to "AM335x_PRU.cmd".

3.2.2.4 Makefile

This file is going to invoke the GCC compiler, to tell it where are the resources needed to compile Hello_PRU.c into the ".out" file.

3.2.2.5 deploy.sh

This is a bash script that is going to clean the project and to call the Makefile. Once the compilation is finished, deploy.sh copy the resulting file ".out" from the "/gen/" folder to into "/lib/firmware/am335x-pru1-fw" folder. This last folder is very important, when the PRU1 is kicked off, it is going to execute the ".out" file placed in this folder (the corresponding folder for PRU0 is /lib/firmware/am335x-pru0-fw).

3.2.3 Running the example

Now, everything is ready to test the PRU setup, you just have to go in the "Hello_PRU" folder and enter the command:

```
sh deploy.sh
```

The "*deploy.sh*" script is run, calls the "*MAKEFILE*", places the result of the compilation and kicks of the PRU. Finally, the LED should be blinking on PIN P8_45.

Chapter 4

RPMsg

The next step is to enable the communication between the PRUs and the ARM®Cortex. This will be very useful when it comes to send data collected with the PRUs.

The different steps are based on the work of [Raven, 2016], who presents a very complete set of instructions in order to enable the RPMsg framework in his project: *Using the Beaglebone™ Green Programmable Real-Time Unit with the Remoteproc and Remote Messaging Framework to Capture and Play Data from an ADC*.

4.1 Presentation of RPMsg

TI explains it better than me:

RPMsg is a message passing mechanism that requests resources through Remoteproc and builds on top of the virtio framework. Shared buffers are requested through the resource_table and provided by the Remoteproc module during PRU firmware loading. The shared buffers are contained inside a vring data structure in DDR memory. There are two vrings provided per PRU core, one vring is used for messages passed to the ARM® and the other vring is used for messages received from the ARM®. System level mailboxes are used to notify cores (ARM® or PRU) when new messages are waiting in the shared buffers.

[TexasInstrument, 2017b]

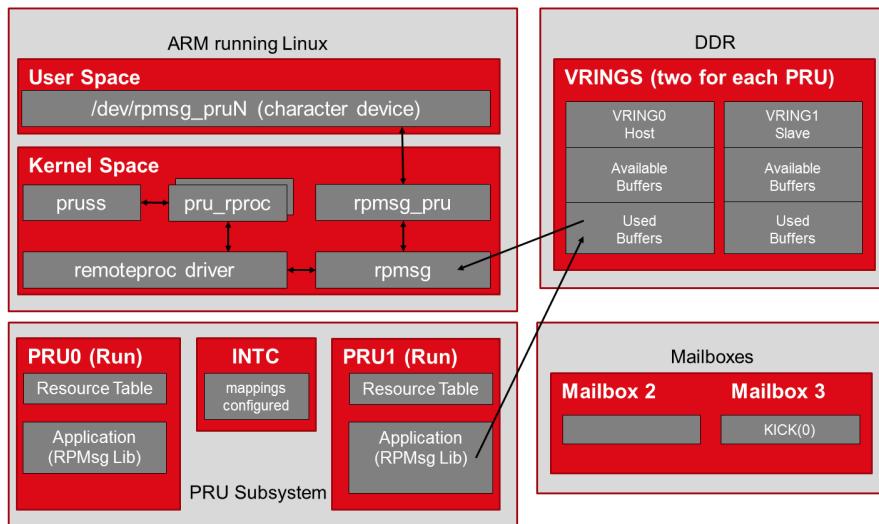


Figure 4.1: Interaction between the ARM® and the PRUs when using RPMsg [TexasInstrument, 2017b]

As explained above, RPMsg uses Remoteproc to transfer messages between the PRUs and the ARM®. Actually, Remoteproc has already been setup in the Chapter 3 when we have loaded the following device tree :

```
uboot_overlay_pru=/lib/firmware/AM335X-PRU-RPROC-4-9-TI-00A0.dtbo
```

Now we are going to enable the RPMsg mechanism.

4.2 Setup

We are going to recompile some device trees:

```
cd /opt/source/bb.org-overlays/  
make  
make install
```

Then a new device tree must be added when we boot the Beaglebone™:

```
cd  
nano /boot/uEnv.txt
```

Go to the section:

```
###Custom Cape
```

and add the following line:

```
###Custom Cape  
dtb_overlay=/lib/firmware/am335x-boneblack.dtbo
```

Then save, quite the file and reboot the BeagleBone™ Black. In order to verify that everything is ready, once the board is on and after few seconds you can go to:

```
cd /sys/bus/platform/devices  
ls
```

In this folder you should be able to see:

```
4a300000.pruss  
4a320000.intc  
4a334000.pru0  
4a338000.pru1
```

If yes, then everything is ok.

4.3 Testing

Now we are going to use the RPMsg framework with a small example in which we are going to send a "Hi PRU" message from the ARM® to the PRU, which is going to answer: "Hi Cortex-A8". Go back to the "/root" folder and create a new folder:

```
cd  
mkdir Test_RPMsg
```

this folder will contain the code for the ARM® a nested folder: "*PRU_codes*", lets create this last folder:

```
cd Test_RPMsg  
mkdir PRU_codes
```

4.3.1 Code for the Cortex-A8

Inside the "Test_RPMsg" folder create these files :

- deploy_echo_ARM.sh
- rpmsg_pru_user_space_echo.c

4.3.1.1 deploy_echo_ARM.sh

It is only a bash script that is going to compile *rpmsg_pru_user_space_echo.c* and execute it.

4.3.1.2 rpmsg_pru_user_space_echo.c

This code is going to be executed by the Cortex-A8, it will open the device character for PRU1, send 10 "Hello PRU!" messages through the RPMsg channel and read the answer into the device character.

```
#define NUM_MESSAGES    10
#define DEVICE_NAME     "/dev/rpmsg_pru31"

int main(void)
{
    struct pollfd pollfds[1];
    int i;
    int result = 0;

    /* Open the rpmsg_pru character device file */
    pollfds[0].fd = open(DEVICE_NAME, O_RDWR);

    /*
     * If the RPMsg channel doesn't exist yet the character device
     * won't either.
     * Make sure the PRU firmware is loaded and that the rpmsg_pru
     * module is inserted.
     */
    if (pollfds[0].fd < 0) {
        printf("Failed to open %s\n", DEVICE_NAME);
        return -1;
    }

    /* The RPMsg channel exists and the character device is opened */
    printf("Opened %s, sending %d messages\n\n", DEVICE_NAME, NUM_MESSAGES);

    for (i = 0; i < NUM_MESSAGES; i++) {
        /* Send 'hello world!' to the PRU through the RPMsg channel */
        result = write(pollfds[0].fd, "hello PRU!", 10);
        if (result > 0)
            printf("Message %d: Sent to PRU\n", i);

        /* Poll until we receive a message from the PRU and then print it */
        result = read(pollfds[0].fd, readBuf, 13);
        if (result > 0)
            printf("Message %d received from PRU:%s\n\n", i, readBuf);
    }

    /* Received all the messages the example is complete */
    printf("Received %d messages, closing %s\n", NUM_MESSAGES, DEVICE_NAME);

    /* Close the rpmsg_pru character device file */
    close(pollfds[0].fd);

    return 0;
}
```

Figure 4.2: Main loop of the rpmsg_pru_user_space_echo code

4.3.2 Code for the PRU

Then you will put a file and 4 folders into the "*PRU_codes*" folder, those codes are going to be executed on PRU0 and PRU1:

- `deploy_echo.sh`
- the "*lib*" folder which contains some needed libraries
- the "*include*" folder which contains resources files for the different TI processors
- `PRU_Halt` which contains all needed codes for PRU0:
 - `AM335x_PRU.cmd`
 - `main.c`
 - `Makefile`
 - `resource_table_empty.h`
- `PRU_RPMsg_Echo_Interrupt1`, which contains the codes for PRU1:
 - `AM335x_PRU.cmd`
 - `main.c`
 - `Makefile`
 - `resource_table_1.h`

4.3.2.1 `deploy.sh`

As for the Cortex-A8 folder, this is a bash script that computes the codes for both PRU and that launches them.

4.3.2.2 `PRU_Halt`

In order to avoid any problems we are going to stop the PRU0 as soon as we started it, this is the role of the "`__Halt()`" function in `main.c` provided by [TexasInstrument, 2014] in the Software Support Package.

```
33
34 #include <stdint.h>
35 #include "resource_table_empty.h"
36
37 int main(void)
38 {
39     __halt();
40 }
41
```

Figure 4.3: Main loop of the `PRU_Halt` code, "`__Halt()`" function stops PRU0

4.3.2.3 `PRU_RPMsg_Echo_Interrupt1`

This is the interesting part of the PRU codes, as for the section 3.4, we need the "`AM335x_PRU.cmd`" and "`resource_table_1.h`" and a "`Makefile`". The `main.c` code is presented in figure 4.4.

After creating the device character "`rpmmsg_pru31`" for the communication with the Cortex-A8, the PRU is going to wait for receiving a message from the Cortex. Each time it receives a message, the PRU is going to send back a message "`Hello_Cortex-A8!`" using the `pru_rpmsg_send()` function.

```

#define VIRTIO_CONFIG_S_DRIVER_OK 4

uint8_t payload[RPMMSG_BUF_SIZE];

/*
 * main.c
 */
void main(void)
{
    struct pru_rpmsg_transport transport;
    uint16_t src, dst, len;
    volatile uint8_t *status;

    /* Allow OCP master port access by the PRU so the PRU can read external memories */
    CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

    /* Clear the status of the PRU-ICSS system event that the ARM will use to 'kick' us */
    CT_INTC.SICR_bitSTS_CLR_IDX = FROM_ARM_HOST;

    /* Make sure the Linux drivers are ready for RPMsg communication */
    status = &resourceTable.rpmsg_vdev.status;
    while (!(status & VIRTIO_CONFIG_S_DRIVER_OK));

    /* Initialize the RPMsg transport structure */
    pru_rpmsg_init(&transport, &resourceTable.rpmsg_vring0, &resourceTable.rpmsg_vring1, TO_ARM_HOST, FROM_ARM_HOST);

    /* Create the RPMsg channel between the PRU and ARM user space using the transport structure. */
    while (pru_rpmsg_channel(RPMMSG_NS_CREATE, &transport, CHAN_NAME, CHAN_DESC, CHAN_PORT) != PRU_RPMMSG_SUCCESS);
    while (1) {
        /* Check bit 30 of register R31 to see if the ARM has kicked us */
        if (_R31 & HOST_INT) {
            /* Clear the event status */
            CT_INTC.SICR_bitSTS_CLR_IDX = FROM_ARM_HOST;
            /* Receive all available messages, multiple messages can be sent per kick */
            while (pru_rpmsg_receive(&transport, &src, &dst, payload, &len) == PRU_RPMMSG_SUCCESS) {
                /* Echo the message back to the same address from which we just received */
                strcpy((char *)payload, "Hello Cortex-A8!");
                pru_rpmsg_send(&transport, dst, src, payload, 16);
            }
        }
    }
}

```

Figure 4.4: Main loop of the PRU_RPMsg_Echo_Interrupt1code

4.3.3 Starting the project

Once you have placed every file in the Test_RPMsg folder you can start both PRUs and Cortex-A8, for this, go into the PRU_codes folder and execute the deploy_echo.sh script:

```

cd
cd /Test_RPMsg/PRU_codes
sh deploy_echo.sh

```

The go into the Test_RPMsg folder and execute the other bash script:

```

cd
cd /Test_RPMsg
sh deploy_echo_ARM.sh

```

You should see something like in figure 4.5 in the console.

If the both examples of sections 3 and 4.1 were run successfully then you are good to go.

```
[root@beaglebone:~/Test_RPMsg# sh deploy_echo_ARM.sh
-----#####---Compiling C code-----#####
-----#####---Starting...
Opened /dev/rpmsg_pru31, sending 10 messages

Message 0: Sent to PRU
Message 0 received from PRU:Hello Cortex-A8!

Message 1: Sent to PRU
Message 1 received from PRU:Hello Cortex-A8!

Message 2: Sent to PRU
Message 2 received from PRU:Hello Cortex-A8!

Message 3: Sent to PRU
Message 3 received from PRU:Hello Cortex-A8!

Message 4: Sent to PRU
Message 4 received from PRU:Hello Cortex-A8!

Message 5: Sent to PRU
Message 5 received from PRU:Hello Cortex-A8!

Message 6: Sent to PRU
Message 6 received from PRU:Hello Cortex-A8!

Message 7: Sent to PRU
Message 7 received from PRU:Hello Cortex-A8!

Message 8: Sent to PRU
Message 8 received from PRU:Hello Cortex-A8!

Message 9: Sent to PRU
Message 9 received from PRU:Hello Cortex-A8!

Received 10 messages, closing /dev/rpmsg_pru31
root@beaglebone:~/Test_RPMsg# ]
```

Figure 4.5: Expected result for the Test_RPMsg folder

Appendix A

PIN Header 8

Pin	\$PINS	ADDR	GPIO	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	CPU	Notes
Offset from:														
P8_01														Ground
P8_02		44<10>00	D2ND											Ground
P8_03	6	0x818'016	GPIO6	gpi0[16]										R9 Associated emmc2
P8_04	/	0x819'01C	GPIO7	gpi0[17]										T9 Associated emmc2
P8_05	2	0x808'008	GPIO12	gpi0[12]										R8 Associated emmc2
P8_06	3	0x805'00C	GPIO13	gpi0[13]										T8 Associated emmc2
P8_07	36	0x809'000	GPIO4	gpi0[24]										R7 Associated emmc2
P8_08	37	0x804'094	TIMER4	gpi0[24]										T7 Associated emmc2
P8_09	39		TIMER5	gpi0[25]										T6 Associated emmc2
P8_10	38	0x808'098	TIME6	gpi0[24]										U6 Associated emmc2
P8_11	13	0x804'034	GPIO13	gpi0[13]	pr1_pmu_3u_30_15									R12 Associated HDMI
P8_12	12	0x803'030	GPIO12	gpi0[12]	pr1_pmu_3u_30_14									T12 Associated HDMI
P8_13	9	0x804'024	EHRRWMB	gpi0[23]										T10 Associated HDMI
P8_14	10	0x804'020	GPIO10	gpi0[26]										T11 Associated HDMI
P8_15	15	0x803'03C	GPIO15	gpi0[15]	pr1_pmu_3u_31_15									U13 Associated HDMI
P8_16	14	0x803'038	GPIO14	gpi0[14]	pr1_pmu_3u_31_14									U13 Associated HDMI
P8_17	11	0x802'020	GPIO27	gpi0[27]										V13 Associated HDMI
P8_18	35	0x808'08C	GPIO21	gpi0[21]	mac00_ts									U12 Associated HDMI
P8_19	8	0x802'020	GPIO22	gpi0[22]										U10 Associated HDMI
P8_20	33	0x804'084	GPIO31	gpi0[31]	pr1_pmu_3u_31_13	pr1_pmu_3u_30_13								V9 Associated emmc2
P8_21	32	0x808'080	GPIO30	gpi0[30]	pr1_pmu_3u_31_12	pr1_pmu_3u_30_12								U9 Associated emmc2
P8_22	5	0x804'014	GPIO4	gpi0[15]										V8 Associated emmc2
P8_23	4	0x804'010	GPIO4	gpi0[14]										U8 Associated emmc2
P8_24	1	0x804'004	GPIO1	gpi0[11]										V7 Associated emmc2
P8_25	0	0x800'000	GPIO0	gpi0[10]										U7 Associated emmc2
P8_26	31	0x807'07C	GPIO29	gpi0[29]										V6 Associated HDMI
P8_27	56	0x800'060	GPIO22	gpi0[22]	pr1_pmu_31_8	pr1_pmu_30_8								U5 Associated HDMI
P8_28	58	0x808'068	GPIO24	gpi0[24]	pr1_pmu_31_10	pr1_pmu_30_10								V5 Associated HDMI
P8_29	57	0x804'064	GPIO23	gpi0[23]	pr1_pmu_31_9	pr1_pmu_30_9								R5 Associated HDMI
P8_30	59	0x804'06C	GPIO25	gpi0[25]	pr1_pmu_31_11	pr1_pmu_30_11								R6 Associated HDMI
P8_31	54	0x808'008	UARTR15	gpi0[10]	uart5_ctsn	uart5_ctsn								V4 Associated HDMI
P8_32	55	0x805'00C	UARTR11	gpi0[11]	uart5_itn	uart5_itn								T5 Associated HDMI
P8_33	53	0x804'004	UARTR17SN	gpi0[9]	uart4_itn	uart4_itn								V3 Associated HDMI
P8_34	51	0x804'00C	UARTR17N	gpi0[7]	uart4_itn	uart4_itn								U4 Associated HDMI
P8_35	50	0x804'000	UARTR14	gpi0[6]	uart4_ctsn	uart4_ctsn								V2 Associated HDMI
P8_36	80	0x808'028	UARTR15_CTSN	gpi0[16]	uart5_ctsn	uart5_ctsn								U3 Associated HDMI
P8_37	48	0x804'000	UARTR15_TDX	gpi0[14]	uart2_ctsn	uart2_ctsn								U1 Associated HDMI
P8_38	49	0x804'004	UARTR15_RXD	gpi0[15]	uart5_ndx	uart5_ndx								U2 Associated HDMI
P8_39	46	0x808'008	GPIO12	gpi0[12]	pr1_pmu_31_6	pr1_pmu_30_6								T3 Associated HDMI
P8_40	47	0x808'00C	GPIO13	gpi0[13]	pr1_pmu_31_7	pr1_pmu_30_7								T4 Associated HDMI
P8_41	44	0x804'000	GPIO10	gpi0[10]	pr1_pmu_31_4	pr1_pmu_30_4								T1 Associated HDMI
P8_42	45	0x804'004	GPIO11	gpi0[11]	pr1_pmu_31_5	pr1_pmu_30_5								T2 Associated HDMI
P8_43	42	0x808'008	GPIO8	gpi0[8]	pr1_pmu_31_2	pr1_pmu_30_2								R3 Associated HDMI
P8_44	43	0x804'000	GPIO9	gpi0[9]	pr1_pmu_31_3	pr1_pmu_30_3								R4 Associated HDMI
P8_45	40	0x804'000	GPIO6	gpi0[6]	pr1_pmu_31_0	pr1_pmu_30_0								U1 Associated HDMI
P8_46	41	0x804'004	GPIO7	gpi0[7]	pr1_pmu_31_1	pr1_pmu_30_1								R2 Associated HDMI
GPIO NO.														
P8_Header	cat_gpios	AUDI+	GPIO NO.	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	CPU	
The BeagleBone Black P8 Header														
EXPLORE THE BEAGLEBONE BLACK™														
TO LEARN AND EXPLORE THE BEAGLEBONE BLACK™														
www.ExploringBeagleBone.com														

Appendix B

PIN Header 9

Pin	SPI_NCS	AUDR	GPIO	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	PinOut	CPU	Notes
Pg_01	44e10000		GND									Ground		
Pg_02	Offset from:		GND									Ground		
Pg_03	44e10800			DC_3.3V								250mA Max Current		
Pg_04				DC_3.3V								250mA Max Current		
Pg_05				VDD_5V								1A Max Current		
Pg_06				VDD_5V								1A Max Current		
Pg_07				SYS_AV								250mA Max Current		
Pg_08				SYS_AV								250mA Max Current		
Pg_10				PWR_BUT								SV Level (collected by PINC)		
Pg_11	28	0x800070	30	UART4_RXD	gpmc[28]	uart4_rxn_mux2	mmci1_sdio	mmci2_sdio	mmci2_sdio	mmci2_sdio	mmci2_sdio	RESET_OUT	A10	
Pg_12	30	0x800078	60	UART4_TXD	gpmc[29]	uart4_txn_mux2	gpmc[30]	gpmc[31]	gpmc[32]	gpmc[33]	gpmc[34]	gpmc_sdio	T17	All GPIOs & SGPIO output
Pg_13	29	0x800074	31	UART4_RXD	gpmc[29]	uart4_txn_mux2	mmci2_sdio	mmci2_sdio	mmci2_sdio	mmci2_sdio	mmci2_sdio	gpmc_sdio	U18	and SGPIO, Break on input
Pg_14	18	0x800048	50	E_HFPWM1A	gpmc[18]	elphyout1_A_mux1	gpmc[48]	gpmc[49]	gpmc[50]	gpmc[51]	gpmc[52]	gpmc_sdio	U17	
Pg_15	16	0x800040	48	E_HFPWM1B	gpmc[19]	elphyout1_B_mux1	gpmc[46]	gpmc[47]	gpmc[48]	gpmc[49]	gpmc[50]	gpmc_sdio	R13	
Pg_16	19	0x80004C	51	E_HFPWM1B	gpmc[19]	elphyout1_B_mux1	gpmc[49]	gpmc[50]	gpmc[51]	gpmc[52]	gpmc[53]	gpmc_sdio	T14	
Pg_17	87	0x805115	5	I2C1_SCL	gpmc[5]	i2c1_scl	pr1_uad0_sdio	pr1_uad0_sdio	pr1_uad0_sdio	pr1_uad0_sdio	pr1_uad0_sdio	elphyout2_Sysce	A16	
Pg_18	86	0x805138	4	I2C1_SDA	gpmc[4]	i2c1_sda	pr1_uad0_sdio	pr1_uad0_sdio	pr1_uad0_sdio	pr1_uad0_sdio	pr1_uad0_sdio	elphyout2_Sysce	B16	
Pg_19	95	0x80507C	13	I2C2_SCL	gpmc[19]	i2c2_scl	pr1_uad0_sdio	pr1_uad0_sdio	pr1_uad0_sdio	pr1_uad0_sdio	pr1_uad0_sdio	elphyout2_Sysce	B22	
Pg_20	94	0x80507E	12	I2C2_SDA	gpmc[20]	i2c2_sda	pr1_uad0_sdio	pr1_uad0_sdio	pr1_uad0_sdio	pr1_uad0_sdio	pr1_uad0_sdio	elphyout2_Sysce	D17	Allocated msg0_pins
Pg_21	85	0x805074	12	UART2_RXD	gpmc[29]	uart2_rxn_mux2	EMU2_sdio	EMU2_sdio	EMU2_sdio	EMU2_sdio	EMU2_sdio	UART1_SDIO	D18	Allocated I2C2
Pg_22	84	0x805010	2	UART2_RXD	gpmc[29]	uart2_rxn_mux2	EMU2_sdio	EMU2_sdio	EMU2_sdio	EMU2_sdio	EMU2_sdio	UART2_SDIO	B17	
Pg_23	17	0x805444	49	UART2_RXD	gpmc[29]	uart2_rxn_mux2	EMU2_sdio	EMU2_sdio	EMU2_sdio	EMU2_sdio	EMU2_sdio	UART2_SDIO	A17	
Pg_24	87	0x805014	15	UART1_RXD	gpmc[19]	uart1_rxn_mux2	pr1_pmu_pmu_31_7	pr1_pmu_pmu_31_7	pr1_pmu_pmu_31_7	pr1_pmu_pmu_31_7	pr1_pmu_pmu_31_7	UART1_SDIO	V14	
Pg_25	107	0x80501C	17	UART3_RXD	gpmc[21]	uart3_rxn_mux2	pr1_pmu_pmu_31_7	pr1_pmu_pmu_31_7	pr1_pmu_pmu_31_7	pr1_pmu_pmu_31_7	pr1_pmu_pmu_31_7	UART3_SDIO	D15	
Pg_26	96	0x805010	14	UART1_RXD	gpmc[14]	uart1_rxn_mux2	pr1_pmu_pmu_31_7	pr1_pmu_pmu_31_7	pr1_pmu_pmu_31_7	pr1_pmu_pmu_31_7	pr1_pmu_pmu_31_7	UART1_SDIO	A14	Allocated msg0_pins
Pg_27	105	0x805014	115	GR03_19	gpmc[31]	pr1_pmu_pmu_31_5	pr1_pmu_pmu_30_5	pr1_pmu_pmu_30_5	pr1_pmu_pmu_30_5	pr1_pmu_pmu_30_5	pr1_pmu_pmu_30_5	GR03_19	C13	Allocated msg0_pins
Pg_28	103	0x80501C	113	SP1_CSO	gpmc[31]	pr1_pmu_pmu_31_3	pr1_pmu_pmu_31_3	pr1_pmu_pmu_31_3	pr1_pmu_pmu_31_3	pr1_pmu_pmu_31_3	pr1_pmu_pmu_31_3	eCAP2_in_WW12_out	C12	Allocated msg0_pins
Pg_29	101	0x805014	111	SP1_D0	gpmc[31]	pr1_pmu_pmu_31_1	pr1_pmu_pmu_31_1	pr1_pmu_pmu_31_1	pr1_pmu_pmu_31_1	pr1_pmu_pmu_31_1	pr1_pmu_pmu_31_1	GR03_20	B13	Allocated msg0_pins
Pg_30	102	0x805010	112	SP1_D1	gpmc[31]	pr1_pmu_pmu_31_2	pr1_pmu_pmu_30_2	pr1_pmu_pmu_30_2	pr1_pmu_pmu_30_2	pr1_pmu_pmu_30_2	pr1_pmu_pmu_30_2	GR03_20	D12	Allocated msg0_pins
Pg_31	100	0x805010	110	SP1_SDCK	gpmc[31]	pr1_pmu_pmu_31_0	pr1_pmu_pmu_30_0	pr1_pmu_pmu_30_0	pr1_pmu_pmu_30_0	pr1_pmu_pmu_30_0	pr1_pmu_pmu_30_0	GR03_20	A13	Allocated msg0_pins
Pg_32				VADC								GR03_20		
Pg_33				AN4								C8		
Pg_34				AN6								Ground		
Pg_35				AN5								A6		
Pg_36				AN2								B8		
Pg_37				AN3								B7		
Pg_38				AN0								A7		
Pg_39				AN1								B6		
Pg_40				AN1								C7		
Pg_41A	109	0x805014	20	CLOCK2	gpmc[20]	EMU3_mmu	pr1_pmu_pmu_31_16	time7_mu2	obc042	obc042	obc042	XMM1_event_mru1	D14	Ground
Pg_41B	116	0x805018	20	CLOCK2	gpmc[20]	pr1_pmu_pmu_30_6	emu13	emu13	mcsc1_200	mcsc1_200	mcsc1_200	XMM1_event_mru1	D13	Ground
Pg_42B	89	0x805014	7	SP0_CSO	gpmc[7]	XMM1_event_mru2	mmr2_200	mmr2_200	sp1_sdck	sp1_sdck	sp1_sdck	XMM1_event_mru2	C18	Ground
Pg_43	114	0x805010	114	SP0_D0	gpmc[31]	pr1_pmu_pmu_31_4	pr1_pmu_pmu_30_4	pr1_pmu_pmu_30_4	mcsc1_200	mcsc1_200	mcsc1_200	XMM1_event_mru2	B12	Allocated msg0_pins
Pg_44				GND								Ground		
Pg_45				cat								Ground		
Pg_46				SPIN5								Ground		
Pg_47				AUDR +								Ground		
Pg_48				GND								Ground		
Pg_49				Name								Notes		
												Mode 1		
												Mode 0		
												CPU		

The BeagleBone Black Pg Header

www.ExploringBeagleBone.com

Exploring the BeagleBone Black
Techniques for Designing with Embedded Linux

Bibliography

- AnalogDevices. Digital accelerometer adxl345 data sheet. Technical report, Analogue Devices, 2016.
- Justin Cooper. Device tree overlays. <https://learn.adafruit.com/introduction-to-the-beaglebone-black-device-tree/overview>, 2015.
- Derek Molloy. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*. Wiley, 2014. ISBN 1118935128. URL <http://www.exploringbeaglebone.com/>.
- Gregory Raven. Using the beaglebone green programmable real-time unit with the remoteproc and remote messaging framework to capture and play data from an adc. <https://github.com/Greg-R/pruadc1>, 2016.
- TexasInstrument. Pru software support package. <https://git.ti.com/pru-software-support-package>, 2014.
- TexasInstrument. Pru read latencies. http://processors.wiki.ti.com/index.php/PRU_Read_Latencies, 2017a.
- TexasInstrument. Pru-icss remoteproc and rpmsg. http://processors.wiki.ti.com/index.php/PRU-ICSS_Remoteproc_and_RPMsg, 2017b.
- TexasInstruments. *AM335x and AMIC110 SitaraTM Processors*, 2017.
- Zubeen Tolani. Ptp - programming the prus 1: Blinky. https://www.zeekhuge.me/post/ptp_blinky/, 2016.
- Mark A. Yoder. Ebc exercise 30 pru via remoteproc and rpmsg. https://elinux.org/EBC_Exercise_30_PRU_via_remoteproc_and_RPMsg, 2017.