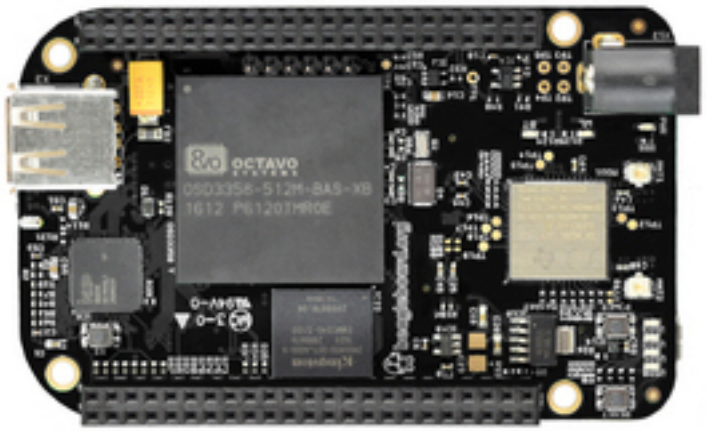
Georgia Tech Mechanical Engineering
The George W. Woodruff School of Mechanical Engineering

Using the PRUs and RPMsg

BeagleBone™ Black Wireless Linux Debian 4.9.45-ti-r57



Pierrick Rauby
Master Thesis Student

Last revision: February 24, 2018

Contents

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 and 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 PRUs : 32-bit microcontrollers*

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

As explained is 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 Ocotovo 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 as Arduino®/Teensy ones. It means that they are able to execute real-time processing, 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, PRUs are very useful when it comes to carry out time critical operations such as fast data acquisitions.

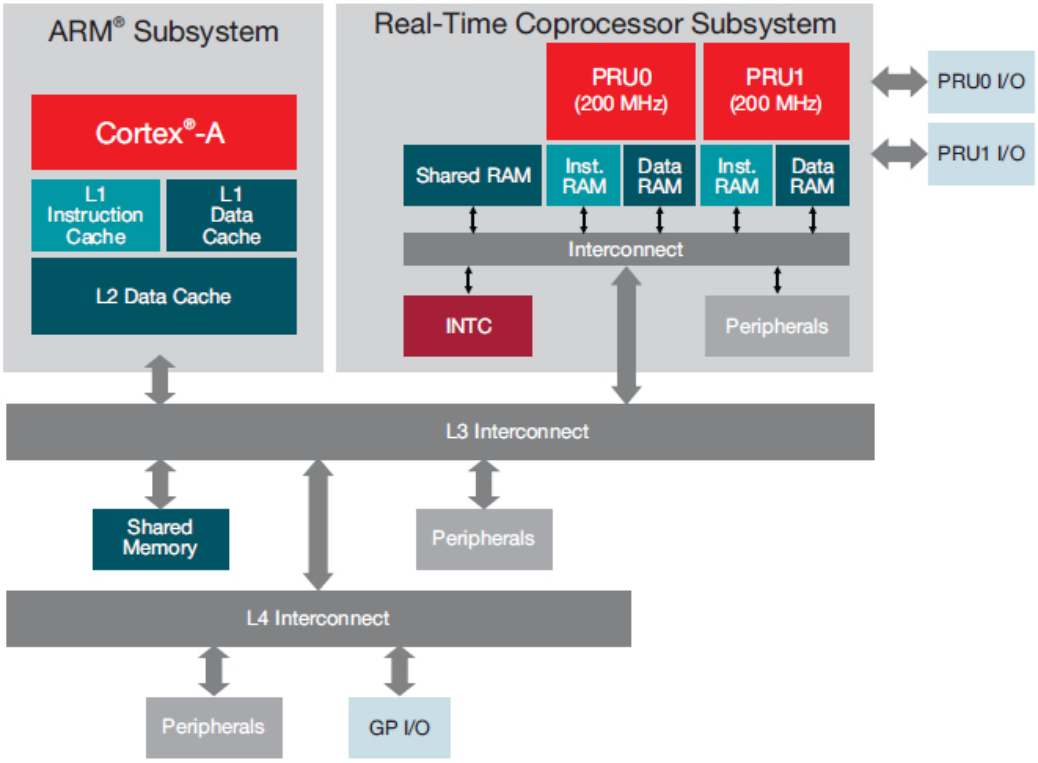


Figure 2.1: Architecture of the AM335x with Cortex®-A8 and the 2 PRUs [TexasInstrument, 2017a]

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 HMDI must be disabled 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 interacts 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 created:

```
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_TXD	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 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 a "hello world!" example in which a small LED is triggered 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 in figure 3.3. Pin P8_45 is used as the output pin and pin P8_1 is connected to the ground of the circuit.

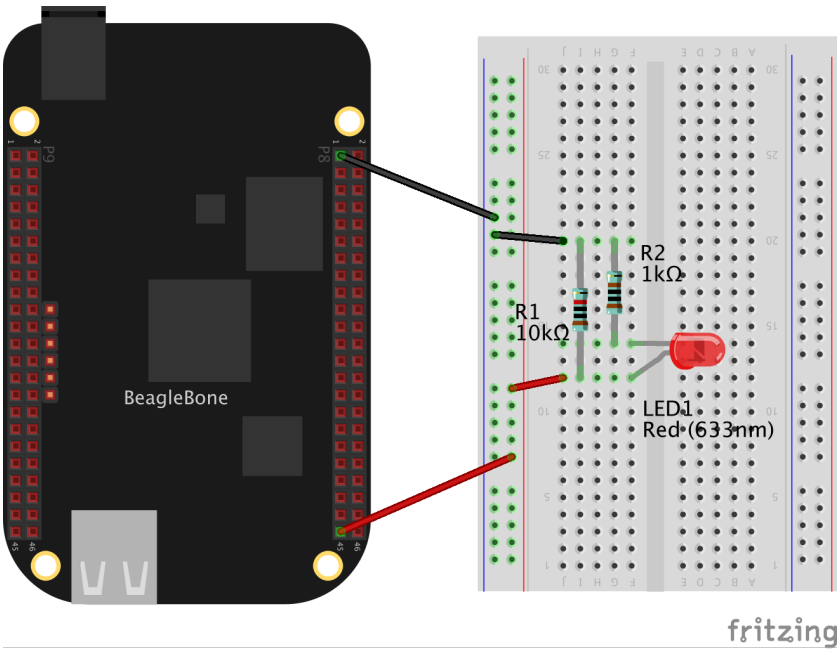


Figure 3.3: The circuit for Hello_PRU program

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 resource files for the different TI processors

3.2.2.1 Hello_PRU.c

This is the C code that 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 s 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 to access any of the 29-R0 registers of the PRU [Tolani, 2016].

```c
#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;

  /* Toggle GPO pins TODO: Figure out which to use */
  gpio = 0x000F;

  /* TODO: Create stop condition, else it will toggle indefinitely */
  while (1) {
    __R30 ^= gpio;
    __delay_cycles(100000000);
  }
}
```

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 on 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 give the location of 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 finish, 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 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]

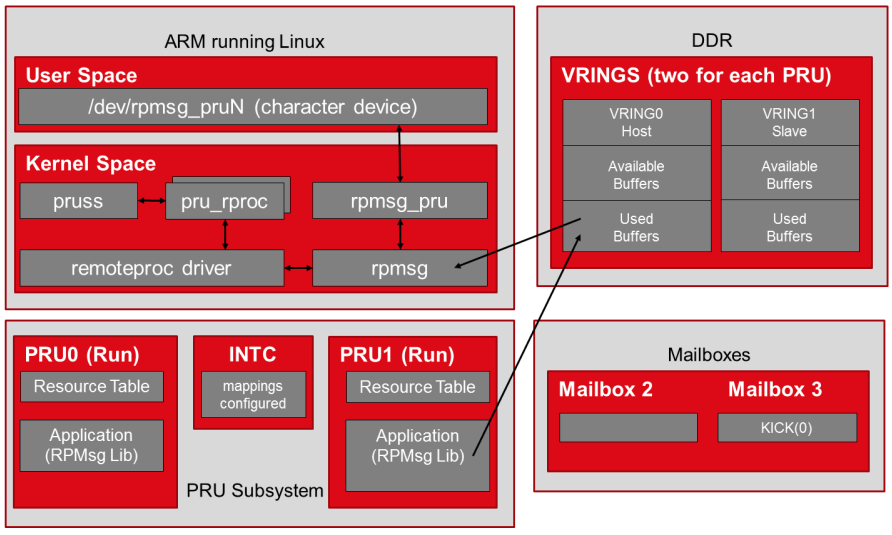


Figure 4.1: Interaction between the ARM®and the PRUs when using RPMsg [TexasInstrument, 2017b]

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, quit 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"*, let's 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.

```c
#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 start 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 *"rpmsg_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.

```c
#define VIRTIO_CONFIG_S_DRIVER_OK 4

uint8_t payload[RPMSG_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_bit.STS_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(RPMSG_NS_CREATE, &transport, CHAN_NAME, CHAN_DESC, CHAN_PORT) != PRU_RPMSG_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_bit.STS_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_RPMSG_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 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
----##########----Compilling 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

EXPLORING BEAGLEBONE
TOOLS AND TECHNIQUES FOR BUILDING WITH EMBEDDED LINUX

The BeagleBone Black P8 Header

www.ExploringBeagleBone.com

Pin	$PINS	ADDR	GPIO	Mode0	Mode1	Mode2	Mode3	Mode4	Mode5	Mode6	Mode7	Name	GPIO	ADDR	$PINS	Pin
P8_01		Offset from:										DGND		44e10800		P8_01
P8_02		44e10800										DGND		0x818/0x18	6	P8_03
P8_03	6	0x818/0x18	38	gpio1[6]	mmc1_dat6	mmc1_dat6					gpio1_6	GPIO1_6	38	0x81c/0x1c	7	P8_04
P8_04	7	0x81c/0x1c	39	gpio1[7]	mmc1_dat7	mmc1_dat7					gpio1_7	GPIO1_7	39	0x808/0x08	2	P8_05
P8_05	2	0x808/0x08	34	gpio1[2]	mmc1_dat2	mmc1_dat2					gpio1_2	GPIO1_2	34	0x80c/0x0c	3	P8_06
P8_06	3	0x80c/0x0c	35	gpio1[3]	mmc1_dat3	mmc1_dat3					gpio1_3	GPIO1_3	35	0x890/0x90	66	P8_07
P8_07	36	0x890/0x90	66	gpio2[2]	gpmc_adv/n_ale	timer4					TIMER4	GPIO2_2	66	0x894/0x94	67	P8_08
P8_08	37	0x894/0x94	67	gpio2[3]	gpmc_oen_ren	timer7					TIMER7	GPIO2_3	67	0x89c/0x9c	69	P8_09
P8_09	39	0x89c/0x9c	69	gpio2[5]	gpmc_be0n_cle	timer5					TIMER5	GPIO2_5	69	0x898/0x98	68	P8_10
P8_10	38	0x898/0x98	68	gpio2[4]	gpmc_wen	timer6					TIMER6	GPIO2_4	68	0x834/0x34	45	P8_11
P8_11	13	0x834/0x34	45	gpio1[13]	mmc1_dat5	mmc1_dat5	mmc2_dat1	eQEP2B_in	pr1_pru0_pru_r30_15		gpio1_13	GPIO1_13	45	0x830/0x30	44	P8_12
P8_12	12	0x830/0x30	44	gpio1[12]	lcd_data18	LCD_DATA19	MMC1_DAT4	EQEP2A_IN	pr1_pru0_pru_r30_14		gpio1_12	GPIO1_12	44	0x824/0x24	23	P8_13
P8_13	9	0x824/0x24	23	gpio0[23]	mmc1_dat1	lcd_data22	mmc2_dat5	ehrpwm2B			EHRPWM2B	GPIO0_26	26	0x828/0x28	47	P8_14
P8_14	10	0x828/0x28	26	gpio0[26]	mmc1_dat2	lcd_data21	mmc2_dat6	ehrpwm2_tripzone_in			GPIO1_15	gpio1[15]	47	0x83c/0x3c	15	P8_15
P8_15	15	0x83c/0x3c	47	gpio1[15]	mmc1_dat7	lcd_data16	mmc2_dat3	eQEP2_strobe	pr1_pru0_pru_r31_15		GPIO1_14	gpio1[14]	46	0x838/0x38	14	P8_16
P8_16	14	0x838/0x38	46	gpio1[14]	mmc1_dat6	lcd_data17	mmc2_dat2	eQEP2_index	pr1_pru0_pru_r31_14		GPIO1_27	gpio1[27]	27	0x82c/0x2c	11	P8_17
P8_17	11	0x82c/0x2c	27	gpio1[27]	mmc1_dat3	lcd_data20	mmc2_dat7	ehrpwm0_synco			GPIO2_1	gpio2[1]	65	0x88c/0x8c	35	P8_18
P8_18	35	0x88c/0x8c	65	gpio2[1]	gpmc_clk_mux0	lcd_memory_clk	gpmc_wait1	mmc2_clk			mcasp0_fsr				8	P8_19
P8_19	8	0x820/0x20	22	gpio0[22]	mmc1_dat8	lcd_data23	mmc1_dat0	mmc2_dat4			EHRPWM2A	gpio0[22]	22	0x824/0x84	63	P8_20
P8_20	33	0x824/0x84	63	gpio1[31]	mmc1_dat9	gpmc_be1n	mmc1_cmd	ehrpwm2B			GPIO1_30	gpio1[30]	62	0x880/0x80	32	P8_21
P8_21	32	0x880/0x80	62	gpio1[30]	mmc1_clk	gpmc_clk	mmc1_clk		pr1_pru1_pru_r31_12	pr1_pru1_pru_r30_12	GPIO1_5	gpio1[5]	37	0x814/0x14	5	P8_22
P8_22	5	0x814/0x14	37	gpio1[5]	mmc1_dat5	gpmc_csn1	mmc1_dat5				GPIO1_4	gpio1[4]	36	0x810/0x10	4	P8_23
P8_23	4	0x810/0x10	36	gpio1[4]	mmc1_dat4	gpmc_csn2	mmc1_dat4				GPIO1_1	gpio1[1]	33	0x804/0x04	1	P8_24
P8_24	1	0x804/0x04	33	gpio1[1]	mmc1_dat1	gpmc_ad1	mmc1_dat1				GPIO1_0	gpio1[0]	32	0x800/0x00	0	P8_25
P8_25	0	0x800/0x00	32	gpio1[0]	mmc1_dat0	gpmc_ad0	mmc1_dat0				GPIO1_29	gpio1[29]	61	0x87c/0x7c	31	P8_26
P8_26	31	0x87c/0x7c	61	gpio1[29]	gpmc_csn0	gpmc_csn0					GPIO2_22	gpio2[22]	86	0x8e0/0e0	56	P8_27
P8_27	56	0x8e0/0e0	86	gpio2[22]	lcd_vsync	gpmc_a8				pr1_pru1_pru_r31_8	GPIO2_24	gpio2[24]	88	0x8e8/0e8	58	P8_28
P8_28	58	0x8e8/0e8	88	gpio2[24]	lcd_pclk	gpmc_a10			pr1_pru1_pru_r30_10	pr1_pru1_pru_r31_10	GPIO2_23	gpio2[23]	87	0x8e4/0e4	57	P8_29
P8_29	57	0x8e4/0e4	87	gpio2[23]	lcd_hsync	gpmc_a9			pr1_pru1_pru_r30_9	pr1_pru1_pru_r31_9	GPIO2_25	gpio2[25]	89	0x8ec/0ec	59	P8_30
P8_30	59	0x8ec/0ec	89	gpio2[25]	gpmc_ad11	lcd_ac_bias_en			pr1_pru1_pru_r30_11	pr1_pru1_pru_r31_11	GPIO2_10	gpio2[10]	74	0x8d8/0d8	54	P8_31
P8_31	54	0x8d8/0d8	74	gpio2[10]	gpmc_a18	lcd_data14	eQEP1_index	mcasp0_axr1	uart5_rxd		GPIO2_11	gpio2[11]	75	0x8dc/0dc	55	P8_32
P8_32	55	0x8dc/0dc	75	gpio2[11]	gpmc_a19	lcd_data15	eQEP1_strobe	mcasp0_ahclkx	uart5_txd		UART5_RTSN	gpio0[11]	11	0x8d4/0d4	9	P8_33
P8_33	53	0x8d4/0d4	9	gpio0[9]	gpmc_a17	lcd_data13	eQEP1B_in	mcasp0_fsr	uart4_rtsn		UART4_RTSN	gpio0[9]	9	0x8cc/0cc	81	P8_34
P8_34	51	0x8cc/0cc	81	gpio2[17]	gpmc_a16	lcd_data12	ehrpwm1B	mcasp0_ahclkr	mcasp0_axr2	uart3_rtsn	UART3_RTSN	gpio0[17]	17	0x8d0/0d0	8	P8_35
P8_35	52	0x8d0/0d0	8	gpio0[8]	gpmc_a14	lcd_data10	ehrpwm1A	mcasp0_axr0	uart4_ctsn		UART4_CTSN	gpio0[8]	8	0x8d8/0d8	80	P8_36
P8_36	50	0x8d8/0d8	80	gpio2[16]	gpmc_a13	lcd_data9	ehrpwm1_tripzone_in	mcasp0_aclkx	uart3_ctsn		UART3_CTSN	gpio2[16]	80	0x8d4/0d4	78	P8_37
P8_37	48	0x8d4/0d4	78	gpio2[14]	gpmc_a12	lcd_data8	ehrpwm0_synco	mcasp0_fsx	uart2_ctsn		UART5_TXD	gpio2[14]	78	0x8d0/0d0	76	P8_38
P8_38	49	0x8d0/0d0	76	gpio2[12]	gpmc_a15	lcd_data11	ehrpwm0B	mcasp0_axr3	uart2_rtsn		UART5_RXD	gpio2[15]	79	0x8d4/0d4	77	P8_39
P8_39	46	0x8d8/0d8	77	gpio2[13]	gpmc_a0	lcd_data6	eQEP2_index	mcasp0_fsx	uart1_rxd	pr1_pru1_pru_r31_6	GPIO2_12	gpio2[12]	76	0x8c8/0c8	46	P8_40
P8_40	47	0x8c8/0c8	76	gpio2[12]	gpmc_a7	lcd_data7	eQEP2_strobe	mcasp0_axr3	uart1_txd	pr1_pru1_pru_r31_7	GPIO2_13	gpio2[13]	77	0x8cc/0cc	47	P8_41
P8_41	44	0x8cc/0cc	77	gpio2[13]	gpmc_a4	lcd_data4	eQEP2A_in	mcasp0_axr2	uart1_ctsn	pr1_pru1_pru_r31_4	GPIO2_10	gpio2[10]	74	0x8b0/0b0	45	P8_42
P8_42	45	0x8b0/0b0	74	gpio2[10]	gpmc_a5	lcd_data5	eQEP2B_in	mcasp0_axr0	uart1_rtsn	pr1_pru1_pru_r31_5	GPIO2_11	gpio2[11]	75	0x8b4/0b4	42	P8_43
P8_43	42	0x8b4/0b4	75	gpio2[11]	gpmc_a2	lcd_data3	ehrpwm2_tripzone_in	mcasp0_aclkr	uart1_txd	pr1_pru1_pru_r31_2	GPIO2_8	gpio2[8]	72	0x8a8/0a8	43	P8_44
P8_44	43	0x8a8/0a8	72	gpio2[8]	gpmc_a3	lcd_data3	ehrpwm0_synco	mcasp0_fsr	uart1_rxd	pr1_pru1_pru_r31_3	GPIO2_9	gpio2[9]	73	0x8ac/0ac	70	P8_45
P8_45	40	0x8ac/0ac	73	gpio2[9]	gpmc_a0	lcd_data2	ehrpwm2A	mcasp0_axr1	uart1_ctsn	pr1_pru1_pru_r30_0	GPIO2_6	gpio2[6]	70	0x8a0/0a0	41	P8_46
P8_46	41	0x8a0/0a0	70	gpio2[6]	gpmc_a1	lcd_data1	ehrpwm2B	mcasp0_aclkx	uart1_rtsn	pr1_pru1_pru_r30_1	GPIO2_7	gpio2[7]	71	0x8a4/0a4	40	P8_45
P8 Header	cat $PINS	ADDR +	GPIO NO.	Name	Mode 0	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7	Name	GPIO NO.	ADDR +	$PINS

Appendix B

PIN Header 9

Pin	$PINS	ADDR	GPIO	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	CPU	Notes
P9_01		44e10000		GND										Ground
P9_02		Offset from:		GND										Ground
P9_03		44e10800		DC_3.3V										250mA Max Current
P9_04				DC_3.3V										250mA Max Current
P9_05				VDD_5V										1A Max Current
P9_06				VDD_5V										1A Max Current
P9_07				SYS_5V										250mA Max Current
P9_08				SYS_5V										250mA Max Current
P9_09				PWR_BUT										5V Level (pulled up PMIC)
P9_10				SYS_RESETn								RESET_OUT	A10	
P9_11	28	0x870/070	30	UART4_RXD	gpio0[30]	uart4_rxd_mux2		mmc1_sdcd	rmii2_crs_dv	gpmc_csn4	mii2_crs	gpmc_wait0	T17	All GPIOs to 4-6mA output
P9_12	30	0x878/078	60	GPIO1_28	gpio1[28]	mcasp0_aclkr_mux3		gpmc_dir	mmc2_dat3	gpmc_csn6	mii2_col	gpmc_be1n	U18	and approx. 8mA on input.
P9_13	29	0x874/074	31	UART4_TXD	gpio0[31]	uart4_txd_mux2		mmc2_sdcd	rmii2_rxerr	gpmc_csn5	mii2_rxerr	gpmc_wpn	U17	
P9_14	18	0x848/048	50	EHRPWM1A	gpio1[18]	ehrpwm1A_mux1		gpmc_a18	mmc2_dat1	rgmii2_td3	mii2_txd3	gpmc_a2	U14	
P9_15	16	0x840/040	48	GPIO1_16	gpio1[16]	ehrpwm1_tripzone_input		gpmc_a16	mii2_txen	rmii2_tctl	gmii2_txen	gpmc_a0	R13	
P9_16	19	0x84c/04c	51	EHRPWM1B	gpio1[19]	ehrpwm1B_mux1		gpmc_a19	mmc2_dat2	rgmii2_td2	mii2_txd2	gpmc_a3	T14	
P9_17	87	0x95c/15c	5	I2C1_SCL	gpio0[5]			pr1_uart0_txd	ehrpwm0_synci	I2C1_SCL	mmc2_sdwp	spi0_cs0	A16	
P9_18	86	0x958/158	4	I2C1_SDA	gpio0[4]			pr1_uart0_rxd	ehrpwm0_tripzone	I2C1_SDA	mmc1_sdwp	spi0_d1	B16	
P9_19	95	0x97c/17c	13	I2C2_SCL	gpio0[13]		pr1_uart0_rts_n	spi1_cs1	I2C2_SCL	dcan0_rx	timer5	uart1_rtsn	D17	Allocated I2C2
P9_20	94	0x978/178	12	I2C2_SDA	gpio0[12]		pr1_uart0_cts_n	spi1_cs0	I2C2_SDA	dcan0_tx	timer6	uart1_ctsn	D18	Allocated I2C2
P9_21	85	0x954/154	3	UART2_TXD	gpio0[3]	EMU3_mux1		pr1_uart0_rts_n	ehrpwm0B	I2C2_SCL	uart2_txd	spi0_d0	B17	
P9_22	84	0x950/150	2	UART2_RXD	gpio0[2]	EMU2_mux1		pr1_uart0_cts_n	ehrpwm0A	I2C2_SDA	uart2_rxd	spi0_sclk	A17	
P9_23	17	0x844/044	49	GPIO1_17	gpio1[17]	ehrpwm0_synco		gpmc_a17	mmc2_dat0	rgmii2_rxdv	gmii2_rxdv	gpmc_a1	V14	
P9_24	97	0x984/184	15	UART1_TXD	gpio0[15]	pr1_pru0_pru_r31_16	pr1_uart0_txd		I2C1_SCL	dcan1_rx	mmc2_sdwp	uart1_txd	D15	
P9_25	107	0x9ac/1ac	117	GPIO3_21	gpio3[21]	pr1_pru0_pru_r31_7	pr1_pru0_pru_r30_7	EMU4_mux2	mcasp1_axr1	mcasp0_axr3	eQEP0_strobe	mcasp0_ahclkx	A14	Allocated mcasp0_pins
P9_26	96	0x980/180	14	UART1_RXD	gpio0[14]	pr1_pru1_pru_r31_16	pr1_uart0_rxd		I2C1_SDA	dcan1_tx	mmc1_sdwp	uart1_rxd	D16	
P9_27	105	0x9a4/1a4	115	GPIO3_19	gpio3[19]	pr1_pru0_pru_r31_5	pr1_pru0_pru_r30_5	EMU2_mux2	mcasp1_fsx	mcasp0_axr3	eQEP0B_in	mcasp0_fsr	C13	Allocated mcasp0_pins
P9_28	103	0x99c/19c	113	SPI1_CS0	gpio3[17]	pr1_pru0_pru_r31_3	pr1_pru0_pru_r30_3	eCAP2_in_PWM2_out	spi1_cs0	mcasp0_axr2	ehrpwm0_synci	mcasp0_ahclkr	C12	Allocated mcasp0_pins
P9_29	101	0x994/194	111	SPI1_D0	gpio3[15]	pr1_pru0_pru_r31_1	pr1_pru0_pru_r30_1	mmc1_sdcd_mux1	spi1_d0		ehrpwm0B	mcasp0_fsx	B13	Allocated mcasp0_pins
P9_30	102	0x998/198	112	SPI1_D1	gpio3[16]	pr1_pru0_pru_r31_2	pr1_pru0_pru_r30_2	mmc2_sdcd_mux1	spi1_d1		ehrpwm0_tripzone	mcasp0_axr0	D12	Allocated mcasp0_pins
P9_31	100	0x990/190	110	SPI1_SCLK	gpio3[14]	pr1_pru0_pru_r31_0	pr1_pru0_pru_r30_0	mmc0_sdcd_mux1	spi1_sclk		ehrpwm0A	mcasp0_aclkx	A13	Allocated mcasp0_pins
P9_32				VADC										1.8 ADC Volt. Ref.
P9_33				AIN4									C8	1.8V input
P9_34				AGND										Ground for ADC
P9_35				AIN6									A8	1.8V input
P9_36				AIN5									B8	1.8V input
P9_37				AIN2									B7	1.8V input
P9_38				AIN3									A7	1.8V input
P9_39				AIN0									B6	1.8V input
P9_40				AIN1									C7	1.8V input
P9_41A	109	0x9b4/1b4	20	CLKOUT2	gpio0[20]	EMU3_mux0	pr1_pru0_pru_r31_16	timer7_mux1	clkout2	tclkin		xdma_event_intr1	D14	Both to P21 of P11
P9_41B		0x9a8/1a8	116	GPIO3_20	gpio3[20]	pr1_pru0_pru_r31_6	pr1_pru0_pru_r30_6	emu3	Mcasp1_axr0		eQEP0_index	mcasp0_axr1	D13	Both to P21 of P11
P9_42A	89	0x964/164	7	GPIO0_7	gpio0[7]	xdma_event_intr2	mmc0_sdwp	spi1_sclk	pr1_ecap0_ecap_capin_apwm_o	spi1_cs1	uart3_txd	eCAP0_in_PWM0_out	C18	Both to P22 of P11
P9_42B		0x9a0/1a0	114	GPIO3_18	gpio3[18]	pr1_pru0_pru_r31_4	pr1_pru0_pru_r30_4		Mcasp1_aclkx	Mcaspo_axr2	eQEP0A_in	Mcasp0_aclkr	B12	Allocated mcasp0_pins
P9_43				GND										- See Pg.50 of the SRM
P9_44				GND										Ground
P9_45				GND										Ground
P9_46	cat		(Mode 7)	GND										Ground
P9	$PINS	ADDR +	GPIO NO.	Name	Mode 7						Mode 1	Mode 0	CPU	Notes

EXPLORINGBEAGLEBONE
TOOLS AND TECHNIQUES FOR BUILDING WITH EMBEDDED LINUX

The BeagleBone Black P9 Header

www.ExploringBeagleBone.com

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.