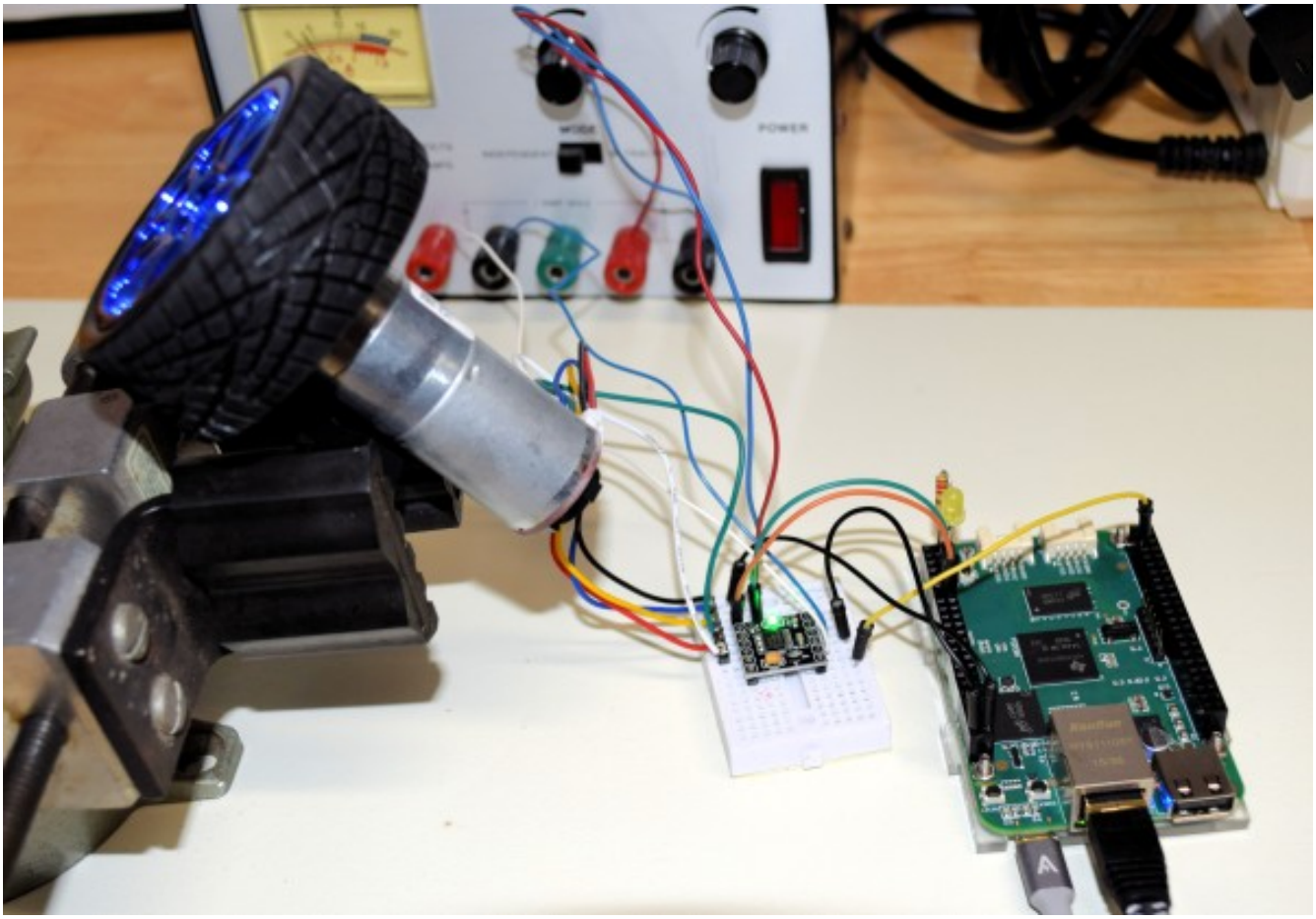

A Motor Speed Control Using Beaglebone Green Programmable Real-Time Unit with the RemoteProc and Remote Messaging Framework

Based on a Project Published by Texas Instruments

Gregory Raven

December 18, 2016

Beaglebone Green PRU PID Motor Speed Control Project

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This is the documentation for an embedded GNU/Linux project utilizing the RemoteProc and RPMsg framework in the Beaglebone Green (BBG) development board. The project repository is located here:

https://github.com/Greg-R/pru-pid-motor

The inspiration for this project came from an example project published by Texas Instruments. The Texas Instruments project is based on "Code Composer Studio", which is an "Integrated Development Environment" (IDE):

http://processors.wiki.ti.com/index.php/PRU_Training:_PRU_PID_Motor_Demo

There is also a PDF file which describes the project in detail:

http://www.ti.com/lit/ug/tidubj6/tidubj6.pdf

The TI project requires a relatively complex cross-compiler installation. This project is designed to be done via SSH terminal connection, and all software compilation was done on the Beaglebone Green target device using the PRU C compiler (clpru). The VIM text editor was used to edit files, however, any text editor available in the Debian distribution can be used.

The Debian-based GNU/Linux distribution used on the BBG can be downloaded from this page:

http://beagleboard.org/latest-images

The “IOT” (non-GUI) image was chosen, as this provides the shortest path to get the project up and running.

Recent developments in the Texas Instruments PRU support include the RemoteProc and Remote Messaging frameworks, as well as an extensively documented C compiler and much additional supporting documentation. This project utilizes these frameworks and is entirely dependent upon C code in both the PRU and GNU/Linux user space. For further information, refer to the detailed examples provided by TI in the “PRU Support Package”:

https://git.ti.com/pru-software-support-package

A listing of additional resources is found in the Resources chapter.

The motor recommended by TI was purchased and tested. However, a better motor with an integrated encoder was found on eBay and is recommended. A chapter is included with describes this motor-encoder and how to obtain one.

1.1 Project Goals

This project demonstrates an electronic speed control for a DC motor which is implemented with the PRUs included with the Beaglebone Green. Beyond its usefulness as a demonstration project, it could be used in a robotics project such as a “mobile robot”.

The basic principle of the speed controller is "Proportional Integral Derivative" feedback control, which is a common feedback controller used in digital systems.

Here is an excellent reference article on PID controllers:

http://www.wescottdesign.com/articles/pid/pidWithoutAPhd.pdf

1.2 Limitations

All of the development was done as root user via ssh on the BeagleBone Green. This is generally not a good practice, however, considering this as an embedded and experimental project it was not considered to be a serious drawback.

No attempt was made to optimize the response of the PID controller. The default values for the PID controller create a stable loop with the recommended DC motor-encoder. Optimization will depend on the particular motor-encoder chosen and this task is left to the interested experimenter.

Chapter 2

System Diagrams

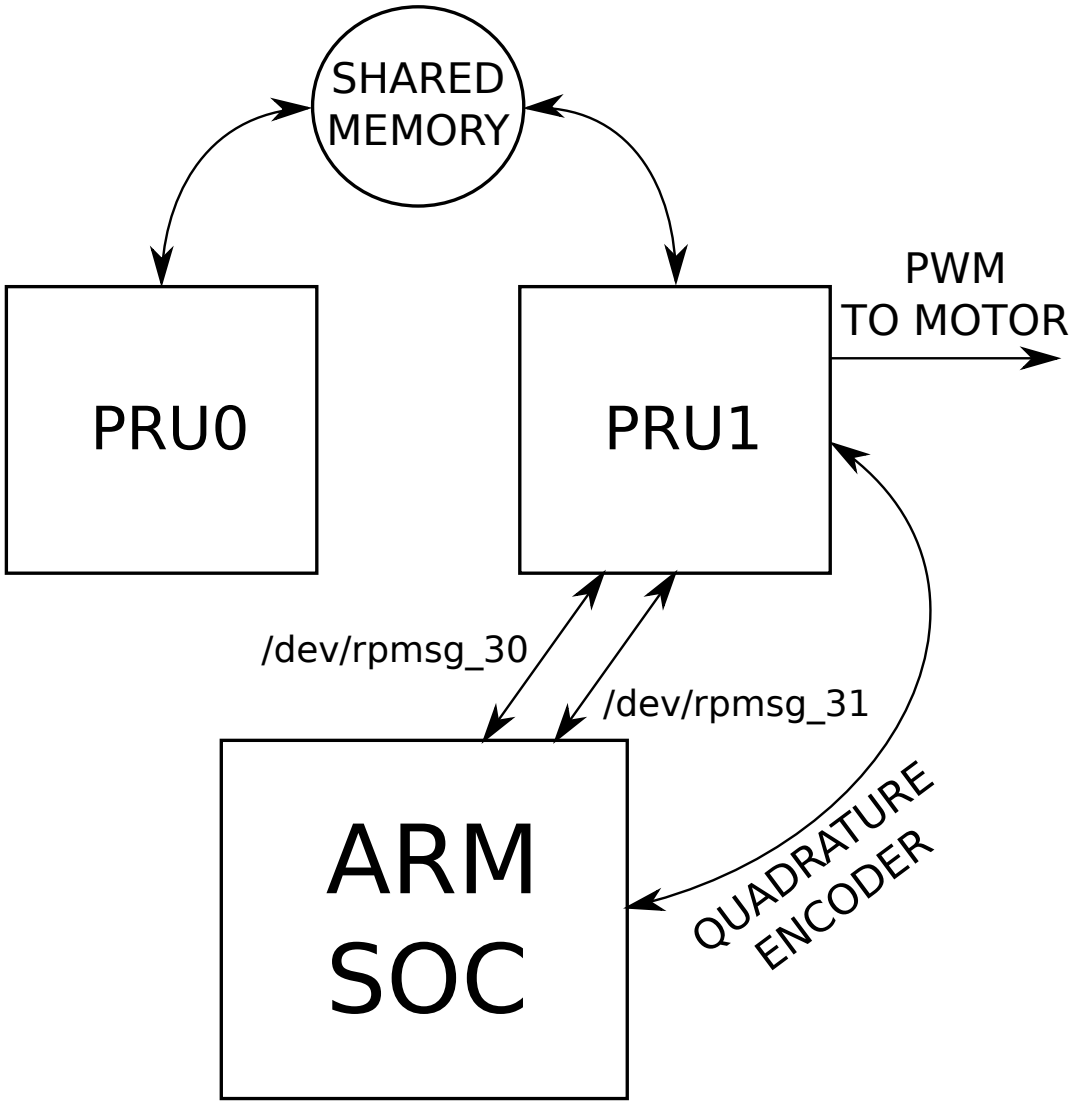

Figure 2.1: PRU PID System on BeagleBone Green AM335X “System On Chip”

The above diagram shows the data pathways used in the project.

2.1 Programmable Real-time Units (PRU)

The system uses both PRUs available on the Beaglebone Green:

1. PRU0 implements the PID controller. The C code in file PRU_PID_0.c contains the math required for the PID controller. The controlled quantity is the DC motor RPM which is obtained from the Quadrature Encoder.

   PRU0 gets performs calculations based on data retrieved (and written to) PRU shared memory.

2. PRU1 acts as the master communicator for the system. PRU1 instantiates two "character devices" via the RemoteProc Messaging framework. These character devices are used for two-way communication between the PRUs and Linux user-space. In addition to the supervisory control function, the PWM module on PRU1 is used to drive the DC motor driver IC.

   The Quadrature encoder used is not part of the PRU-ICSS. However, the PRUs have access to the encoders which are part of the AM335X "System On Chip" (SOC). Access to this encoder is enabled in PRU1 code.

It is worth noting that the PRUs each have a copy of the same “PWMSS” (Pulse-Width Modulation Subsystem) as the ones located in the AM335X SOC. However, only a single pin is connected from the PRU-ICSS to the outside world. Since this single pin is used for PWM, the decoder-counter function must be done outside the PRU-ICSS thus one of the SOC’s eQEP decoder-counters is used and is accessed by PRU1 via the Interface/OCP Master port.

The PWMSS sub-system is surprisingly complex. A detailed description of the features of this sub-system is available in the AM335x Technical Reference Manual:

http://www.ti.com/lit/ug/spruh73o/spruh73o.pdf

2.2 PRU Shared Memory

The firmwares running on the two PRUs use a C structure which is placed into "shared memory". The shared memory is a feature built into the PRU-ICSS architecture. The shared memory is allocated by an entry in the "linker command file" which is included in the Github repository.

The structure in shared memory allows the two PRUs to synchronize the PID control parameters. PRU1 receives the parameters via user-space via the character devices and then writes them to shared memory. This makes the parameters available to PRU0, which is then able to perform PID control loop calculations. In turn, PRU0 calculates the PWM duty cycle, and this is written to shared memory. PRU1 then applies this to the PWM output.

PRU1 is responsible for reading the Quadrature Encoder, which is accessed via the OCP bus master. PRU1 writes this data to shared memory, which enables PRU0 to access this information for PID loop calculations.

A mystery is how the PRUs are able to share memory without locking. It is possible for the PRUs to simultaneously read and write to the same memory location without introducing errors?

Chapter 3

Motor-Encoder

The TI project recommends using this motor-encoder:

https://www.sparkfun.com/products/13260

While this motor-encoder kit can be made to work, a better solution was found on eBay:

Figure 3.1: eBay Motor-encoder

The eBay motor-encoder is a nicely constructed geared DC motor and it has an integrated electronic encoder using Hall-Effect devices.

Also includes is a solid mounting bracket, a wired connector, a brass bushing and a matching wheel. For the purposes of using this in a mobile robot this is very good indeed!

The seller of this motor-encoder was “kanzezol” as of December 2016. This seller also has what appears to be a variant motor-encoder which is less expensive, but it does not include the

mounting hardware and wheel.

3.1 Quadrature Encoder

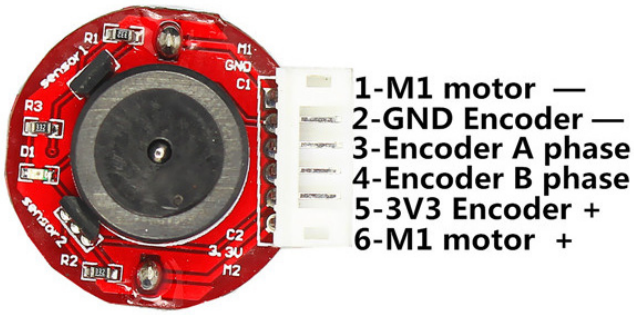

Figure 3.2: The Integrated Quadrature Encoder

The suggested DC motor includes an integrated "Quadrature Encoder". The Quadrature Encoder consists of two Hall-Effect sensors and a rotating magnet attached to the motor shaft. The rotating magnet has areas of alternating north and south poles. Thus as the motor shaft turns, the Hall-Effect sensors output a series of pulses with frequency proportional to the rotation velocity "Revolutions Per Minute" (RPM).

Since there are two Hall-Effect sensors, there are two separate pulse outputs. The Sensors are arranged in physical locations around the rotating magnet such that the two pulse trains overlap. They overlap with a phase difference of approximately 90 degrees, thus the term "quadrature" is used, however, precise 90 degree offset does not appear to be even remotely critical!

The overlapping pulses allow the sensing of rotational direction, since one sensor will fire first in one direction, and the other sensor will fire first in the opposite direction. Direction reversal was not included in this project and the feature was not used.

The encoder requires a 3.3 Volt power source.

The 3.3 Volt bias source for the encoder was found to be somewhat critical. This bias supply could probably be sourced from the BBG, however, this was not attempted.

Chapter 4

PRU Firmware and User-space Program

The “PRU Firmware” are two binary files which are placed in the directory /lib/firmware. These files must have specific names as follows:

- am335x-pru0-fw
- am335x-pru1-fw

The Makefile includes cp commands to copy the firmwares to the /lib/firmware directory.

4.1 The PID Firmware in PRU0: Digital Feedback Loop (pru0adc.c)

The SPI bus C program roughly follows the PRU assembly code written by Derek Molloy. The C code is compiled to a binary file am335x-pru0-fw. The firmware is loaded into PRU0 automatically by the Remoteproc kernel driver.

The program begins with code sequestered from an example code file in TI’s PRU Software Support Package. This codes establishes the character device driver via the “Remote Proc Messenger” kernel driver. There are several lines of RPMsg “set-up” code which appear at the top of the file.

Here is the path to the file from the root directory of the PRU Software Support Package:

`pru-software-support-package/examples/am335x/PRU_RPMsg_Echo_Interrupt0`

There is a sort of “priming” process required whereby a user space program writes to the device driver. The initializes the character driver, which allows it to write data from the PRU to the character driver and thus making the data available in user-space. This is the critical code which performs this function:

```
//  This section of code blocks until a message is received from ARM.
while (pru_rpmsg_receive(&transport, &src, &dst, payload, &len) !=
PRU_RPMSG_SUCCESS) {
}
```

This empty while-loop continues until the user-space code writes a message to the character driver. Upon receipt of a message, the data transport channel is ready to go, and the program breaks out of the while-loop.

After the initialization is complete, the program enters a for-loop. The SPI bus is implemented inside this for-loop. This is done by "bit-twiddling" of register __R30, which is 32 bits in length. The individual bits in __R30, in turn determine the "high" or "low" state at the GPIO, and thus the header pins. The GPIO multiplexing is set via the "Universal IO", which is described in a later chapter.

The code utilizes timing delays and sequential setting and unsetting of bit values of __R30. This is done per the requirements shown in the MCP3008 ADC data sheet. Each pass through the for-loop configures the ADC, and then captures a single 10-bit sample from the ADC.

The top of the for-loop is blocked by a while "polling" loop. The operand of the while-loop is the value of a PRU shared memory location. The PRU1 timing clock code writes to this location at precisely timed intervals. When the while loop detects that three of the bits change from 0s to 1s, the while loop is broken and the SPI bus data acquisition sequence begins. This action is what determines the 8 kHz data sampling rate.

The samples are accumulated in a buffer (int16_t payload[256]). When the buffer is filled, the data is written to the character device via a function provided by the RPMsg kernel driver. Here is the code:

```
//  Send frames of 245 samples.
//  The entire buffer size of 512 can't be used for
//  data.  Some space is required by the "header".
//  The data is offset by 512 and then multiplied
//  to make appropriately scaled 16 bit signed integers.
payload[dataCounter] = 50 * ((int16_t)data - 512);
dataCounter = dataCounter + 1;

if (dataCounter == 245) {
pru_rpmsg_send(&transport, dst, src, payload, 490);
dataCounter = 0;
}
```

The dataCounter variable is incremented at the end of each pass through the for loop. When the dataCounter hits 245, the pru_rpmsg_send command is used to write data to the character device. The dataCounter variable is reassigned to zero and the process repeats.

The maximum buffer size which can be handled by RPMsg is 512 bytes (or 256 16 bit integers). However, not all of the buffer can be used as there is a "header" included which takes a few bytes. The number of samples transmitted, 245, was determined empirically. Some study of the kernel drivers needs to be accomplished to better understand this limitation.

Timings are critical, and this was accomplished by using the compiler intrinsic __delay_cycles. Each delay is an absolute value of 5 nanoseconds. This scheme has sufficient timing precision to implement the SPI bus in real time.

4.2 The Firmware in PRU1: PID Control (pru1adc.c)

Chapter 5

RemoteProc and RPMsg Framework

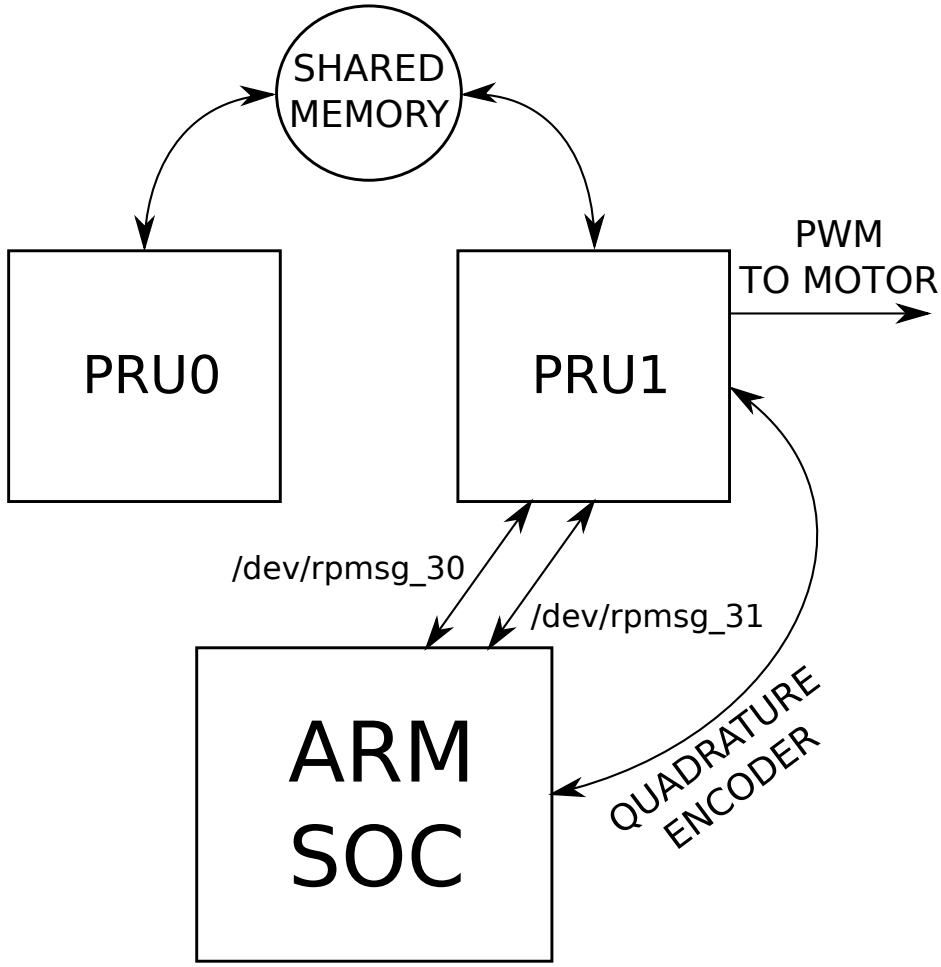

Figure 5.1: PRU<->ARM Character Devices

TI has provided example code and kernel drivers for the "RemoteProc and RemoteProc Messaging Framework". A detailed explanation of this framework is available here:

http://processors.wiki.ti.com/index.php/PRU-ICSS_Remoteproc_and_RPMsg

This framework provides a means of controlling and communicating with the PRUs from user-space, and this project is totally dependent on these functions.

The Remoteproc framework automatically does the job of loading the PRU firmwares from user-space into the PRUs. Via a sysfs entry, the PRUs can be started and halted from the command line. These functions are described in the chapter "Shell Scripts".

The examples provided in the PRU Software Support Package show how to use provided functions to send and receive data from PRU to ARM or ARM to PRU. This is done via character devices which appear in the usual /dev directory. The standard POSIX functions read/write/open/close work with these character devices. This allows for typical systems programming technique to become applicable when working with the PRUs.

This project requires the use of one character device for each PRU. The character driver for PRU0 is the “data stream” for PCM data read from the ADC via the SPI bus. The character device for PRU1 is used to activate the PRU1 Timing Clock as the last action after all systems are initialized. A simple signal is transmitted from user-space to PRU1, and this begins the flow of data through the system.

This project did not require modifications to the loadable kernel modules in the Remoteproc framework. The modules provided with the support package were used as-is.

5.1 The Remoteproc and RPMsg Kernel Modules

“Loadable Kernel Modules” (LKMs) must be active for this project to function. At the shell command line, execute this command:

```
lsmod
```

This will list the LKMs currently loaded. The modules associated with Remoteproc are:

- pruss
- pru_rproc
- pruss_intc

There are two modules associated with RPMsg, and these will appear in the list only after the firmwares are loaded into the PRUs:

```
virtio_rpmsg_bus
rpmsg_pru
```

The Remoteproc kernel modules may not be loaded at boot (depending on boot configuration). However, they can be loaded (or unloaded) anytime after boot with the following commands:

```
modprobe pru_rproc
rmmod pru_rproc
```

“modprobe” is the load command; rmmod is the remove command.

Perhaps a better method of controlling the Remoteproc framework is via the sys virtual file system. A set of shell scripts is included in the repository which includes these commands in the "prumodout" and "prumodin" scripts.

5.2 Files Associated with RemoteProc in the Compilation Process

There are some interesting files in the root directory of the github repository for this project:

- resource_table_0.h
- resource_table_1.h
- AM335x_PRU.cmd

These files were copied verbatim from the PRU Support Package.

Jason Reeder of Texas Instruments has provided an explanation of the files required to compile firmwares for the PRUs:

> There are four files needed in order to build a C project for the PRU using TI's C compiler. Each of the examples and labs in the pru-software-support-package include these files:
>
> 1. yourProgramFile.c
>   This is your C program that you are writing for the PRU.
> 2. AM335x_PRU.cmd
>   This is a command linker file. This is the way that we describe the physical memory map, the constant table entries, and the placement of our code and data sections (into the physical memory described at the top of the file) to the PRU linker. There are some neat things that can be done as far as placing code and data in exact memory locations using this file, but for the majority of projects, this file can remain unchanged.
> 3. resource_table_*.h
>   This file will create a header in the elf binary (generated .out file) that is needed by the RemoteProc Linux driver while loading the code into the PRUs. For the examples in the pru-software-support-package, there are two types of resources that the PRU can request using the resource_table_*.h file:
>   interrupts - letting RemoteProc configure the PRU INTC interrupts saves code space on the PRUs.
>   vrings - requesting vrings in the resource_table file is necessary if rpmsg communication is desired (since the ARM/Linux needs to create the vrings in DDR and then notify the PRU where the vrings were placed) even if no resources are needed, the RemoteProc Linux driver expects the header to exist. Because of this, many examples in the package contain an empty resource_table header file (resource_table_empty.h)
> 4. Makefile

Makefile to build your PRU C program either on the target, on your Linux machine, or even on a Windows machine. The comment at the top of the Makefile tries to explain the environment variable needed for a successful build and how to set it on each of the three supported build development environments.

Chapter 6

Universal IO and Connecting PRU to the Outside World

This project did not require a custom “Device Tree Overlay”. Instead, the “Universal IO” driver was used along with a simple shell script.

The Universal IO project is located at this Github repository:

https://github.com/cdsteinkuehler/beaglebone-universal-io

The configuration is as follows:

```
config-pin P8.30 pruout
config-pin P9.31 pruout
config-pin P9.27 pruout
config-pin P9.29 pruout
config-pin P9.28 pruin
config-pin P9.30 pruout
```

The above can also be put into a file, for example “pru-config”:

```
P8.30 pruout
P9.31 pruout
P9.27 pruout
P9.29 pruout
P9.28 pruin
P9.30 pruout
```

The command to load the above would be:

config-pin -f pru-config

Note that another step required is create the $SLOTS environment variable and also to set on of the Universal-IO device trees as follows:

```
export SLOTS=/sys/devices/platform/bone_capemgr
echo univ-emmc > $SLOTS/slots
```

6.0.1 The PRU GPIO Spreadsheet

Use “git clone” to download this repository:

https://github.com/selsinork/beaglebone-black-pinmux

The spreadsheet file contained in this repository is pinmux.ods. The LibreOffice suite has a spreadsheet application which will read this file.

This spreadsheet is extremely useful when configuring the PRU or other functions to the Beaglebone pin multiplexer.

Chapter 7

Shell Scripts

There are two very important shell scripts located in the shell_scripts of the git repository.

These scripts are very simple and each contain only a single command.

The commands are described in the notes file from this github repository:

https://github.com/ZeekHuge/BeagleScope

And specifically, this is the path to the notes file:

https://github.com/ZeekHuge/BeagleScope/blob/port_to_4.4.12-ti-r31%2B/docs/current_remoteproc_drivers.notes

The commands are seen in section 2:

```
echo "4a334000.pru0" > /sys/bus/platform/drivers/pru-rproc/unbind
echo "4a334000.pru0" > /sys/bus/platform/drivers/pru-rproc/bind
echo "4a338000.pru1"  > /sys/bus/platform/drivers/pru-rproc/unbind
echo "4a338000.pru1" > /sys/bus/platform/drivers/pru-rproc/bin
```

The above shell commands show how the PRUs can “bind” and “unbind” from the remoteproc driver. These commands are extremely useful and their placement in shell scripts allows them to be easily run at the command line by entering “prumodin” or ”prumodout”.

The shell scripts should be copied to /usr/bin so they will be available in any shell.

Chapter 8

Setting up the Remoteproc PRU and Compiler on the Beaglebone Green

The following describes the simplest possible set-up. Everything was done via the command line, and the vim editor was used extensively to develop the C code and shell scripts.

SSH was used to remotely access the BBG from a 64 bit desktop computer running Ubuntu 14.04.

For reference, here is the link to the TI PRU support package:

https://git.ti.com/pru-software-support-package

The above package can be cloned to the BBG. There is a good set of examples and labs included. The labs are documented here:

http://processors.wiki.ti.com/index.php/PRU_Training:_Hands-on_Labs

Note that the files appropriate for the BBG are in the folders with name am335x.

The Makefiles in the labs and examples were designed to work with a particular set-up which can be easily implemented on the BBG.

The following is a list of recommended steps to prepare a BBG for compiling PRU C files. This process assumes a relatively new SD card image which is loaded with the PRU compiler (clpru) and libraries. Another assumption is that the Remoteproc and RPMsg kernel drivers are included and that they are loaded during the start-up process. This is true for some, but not all, recently published images as of November, 2016.

8.1 Activate Remoteproc PRU and Kernel Modules

The newest Beaglebone Debian distributions do not have the Remoteproc framework activated by default!

The following process activates the framework which includes several loadable kernel modules. This is a prerequisite for the remainder of the setup process.

This process was tested using this image:

bone-debian-8.6-iot-armhf-2016-10-30-4gb.img

The "IOT" (Internet Of Things) image includes the set of tools required to install and compile required software. The image was found at this web site:

http://elinux.org/Beagleboard:BeagleBoneBlack_Debian#microSD.2FStandalone:_.28iot.29_.28BeagleBone.2FBeagleBone_Black.2FBeagleBone_Green.29

The IOT distribution includes very useful scripts in the following directory:

/opt/scripts/tools

The script grow_partition.sh will expand the file system on the microsd card to its full capacity. Running this script is highly recommended before proceeding with this process!

8.2 Activate Remoteproc: Step-by-step Process

1. Write Beaglebone image to micro-sd and expand partition as required.
2. Insert micro-sd into BBG slot, press boot and power buttons and release. Non-flasher images may not require the boot button to be pressed, and the board will boot and run from the SD card.
3. ssh debian@192.168.1.7 (or whatever the IP is set to). If you are using a router with a GUI control application, it may have a display which indicates the board is connected and which IP address has been assigned to it.
4. Execute

   ```
   sudo apt-get update
   ```

5. Execute

   ```
   uname -r
   ```

   to verify kernel version. Please note that the Remoteproc framework is still evolving and it is recommended to verify that the kernel used will work with the PRU support package examples.

   The rest of the set-up will be completed using root access. Execute

   ```
   sudo su
   ```

   and authenticate as required to switch to root user.

6. Clone this repository to a convenient directory:

   ```
   git clone https://github.com/RobertCNelson/dtb-rebuilder
   ```

7. Execute:

   ```
   cd dtb-rebuilder/
   cd src/arm
   ```

8. Find and edit the top of the device tree dts file. For BBG, this is:

   ```
   am335x-bonegreen.dts
   ```

   The only change required is to uncomment a single line in the file:

   ```
   /*   #include "am33xx-pruss-rproc.dtsi"  */
   ```

   Unquote the line as follows:

   ```
   #include "am33xx-pruss-rproc.dtsi"
   ```

   Save and exit.

9. Execute:

   ```
   cd /etc/modprobe.d
   ```

   Create a new file named:

   ```
   pruss-blacklist.conf
   ```

   Add this single line to the file:

   ```
   blacklist uio_pruss
   ```

   Save and exit.

10. cd back to the dtb-rebuilder directory. Execute these commands:

    ```
    make
    make install
    reboot
    ```

To verify that the above process was successful:

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

Now look for the following in the output from the ls command:

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

The appearance of the above entries indicates that the Remoteproc PRU activation process was successful.

8.3 PRU Compiler Setup Process

1. Execute these commands:

   ```
   cd /
   ```

   and then

   ```
   find . -name cgt-pru
   ```

   The path may be something like

   ```
   /usr/share/ti/cgt-pru
   ```

   This is the location of the PRU library and includes. However, the clpru compiler binary is not located in this directory. Run this command:

   ```
   which clpru
   ```

   and the result will be something like:

   ```
   /usr/bin/clpru
   ```

   This is the path to the compiler binary.

   The PRU C compiler needs to find the include and lib directories. Execute the following:

   ```
   cd /
   find . -name cgt-pru
   ```

   This should find the following or similar path:

   ```
   /usr/share/ti/cgt-pru
   ```

   The above is the path to the C Compiler includes and lib directories. The Makefiles in the PRU Support Package look for the compiler binary at this path, so the following changes must be made.

   Execute the following commands (as root):

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

   So now the Makefiles will find the compiler executable in the correct location via the link.

2. Now install the PRU Support Package:

   ```
   cd /home/debian
   git clone git://git.ti.com/pru-software-support-package/pru-software-support-package.g
   ```

   This will clone a copy of the latest pru support package.

3. cd into lab_5 in the package and execute the make command:

```
cd pru-software-support-package/labs/lab_5/solution/PRU_Halt
make
```

This will fail, as the Makefile is looking for environment variable $PRU_CGT. Execute:

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

Now execute the make command again. It should succeed. A new directory "gen" should appear. Add the above export command to the start-up commands (.profile or .bashrc).

4. Execute the following:

```
cd gen
cp PRU_Halt.out am335x-pru0-fw
cp am335x-pru0-fw /lib/firmware
```

This renames the executable binary and copies it to the directory at which Remoteproc expects to find PRU firmwares.

5. Now cd into the PRU_RPMsg_Echo_Interrupt1 directory in the same lab_5. Edit main.c as follows:

```
//#define CHAN_NAME "rpmsg-client-sample"
#define CHAN_NAME "rpmsg-pru"
```

The "CHAN_NAME" define is now set to "rpmsg-pru".

6. Now execute almost the same as #9, except this time the firmware for PRU1 is compiled a copied:

```
make
cd gen
cp PRU_RPMsg_Echo_Interrupt1.out am335x-pru1-fw
cp am335x-pru1-fw /lib/firmware
```

The compilation of both PRU firmwares are complete and they are copied to /lib/firmware.

7. reboot

8. Execute:

```
lsmod
```

These kernel modules should be present in the output:

```
pru_rproc              15431  0
pruss_intc              8603  1 pru_rproc
pruss                  12026  1 pru_rproc
```

Now execute:

```
rmmod pru_rproc
modprobe pru_rproc
```

The rmmod command removes the remoteproc module pru_rproc. The modprobe command re-inserts the same module.

9. `cd /dev`
   `ls`

   Look for rpmsg_pru31 character device file. It will be there!

8.4 Additional Configuration Required to Compile the PRU Remoteproc Project

Some components of the PRU Support Package need to be added to the PRU C compiler directories. The Makefile expects to find these components in these directories.

Execute these commands:

```
cd pru-software-support-package/
cp -r include $PRU_CGT/includeSupportPackage
cp lib/rpmsg_lib.lib $PRU_CGT/lib
```

This should complete the configuration required to run the command successfully.

Chapter 9

Running the Project

In order to run the project and hear audio from the speaker, the following steps must have been completed:

- "make" run in XXXX repository directory. Some warnings or errors may be ignored. Check that the C code files pruadc0.c and pruadc1.c compile and firmware files are copied to /lib/firmware.
- Run command

  `prumodout`

  at the command line. Note that this command may cause some errors or warnings to be emitted. This is normal and depends on the current state of the remoteproc kernel driver.
- Run command

  `prumodin`

  at the command line.
- XXX
- Finally, the user-space program is ready to be started!

  ```
  cd user_space
  ./prumsg s 3000
  ```

If all goes well, the motor should begin turning.

The PRU firmwares will continue to run. To stop them, issue the command

`prumodout`

at the command line, and the PRUs will be halted.

Chapter 10

Resources

Github repository for this project:

https://github.com/Greg-R/pru-pid-motor

The Texas Instruments PRU PID Motor Demonstration project:

http://processors.wiki.ti.com/index.php/PRU_Training:_PRU_PID_Motor_Demo

There is also a PDF file which describes the project in detail:

http://www.ti.com/lit/ug/tidubj6/tidubj6.pdf

Texas Instruments PRU Code Generation tools:

http://software-dl.ti.com/codegen/non-esd/downloads/download.htm#PRU

Beagle Bone Green:

https://www.seeedstudio.com/SeeedStudio-BeagleBone-Green-p-2504.html

The Remoteproc Framework and Remote Messaging:

http://processors.wiki.ti.com/index.php/PRU-ICSS_Remoteproc_and_RPMsg

The BeagleScope project:

https://github.com/ZeekHuge/BeagleScope