Real-time Application in Linux using PRUSS

Anjitha M. Anand
M.Tech – Embedded Systems
NIELIT
Calicut, India

Balu Raveendran
Asst. Professor: AEI
RSET
Cochin, India

Shoukath Cherukat
Scientist/Engineer 'D': Embedded Systems
NIELIT
Calicut, India

Abstract—**This paper proposes the use of Linux OS for real-time applications with the aid of Sitara ARM processor, AM335x. The Sitara AM335x SoC is equipped with two 32-bit low-latency microcontrollers called Programmable Real-time Units (PRUs), forming Programmable Real-time Unit Subsystem (PRUSS). PRUs can be individually programmed to do a specific task independent of the arm processor. This provides many advantages like fast operation with near to real-time processing speed and offload of deterministic and time-critical tasks from the ARM core.**

Keywords—Sitara ARM Processor; PRUSS; PRU; Beaglebone Black; AM335x

I. INTRODUCTION

In this paper, we have proposed the use of Beaglebone Black Rev C [1], based on Sitara Processor AM3358 SoC, for real-time application through the example of measuring ambient temperature of a room in real-time using the on-board PRU which runs independent of the ARM Cortex-A8 processor.

The AM335x SoC can run various operating systems optimized for ARM processors like Linux, Android, RTOS, and Windows Embedded or even without OS using TI's StarterWare. Also, for specific applications, customized kernel can be compiled and used according to requirement. But such a High Level Operating System (HLOS) running on the ARM core cannot guarantee fast response time for real time applications even though the OS itself performs very efficiently on the SoC. This is mainly because a typical HLOS architecture involves several layers of memory or interconnects, which prevents it from providing fast response time even for a simple operation like toggling the status of a GPIO pin configured as digital output. However, the PRU has direct access to GPIO pins which ensures that such applications are completed in the least time possible [2].

II. PROGRAMMABLE REAL-TIME UNIT SUB SYSTEM (PRUSS)

The PRUSS consists of two real-time cores which are independently programmable and are capable of running at 200MHz. Each core can control 16 GPIO pins. The AM335x SoC can be considered to have three independent cores. The PRUSS can operate in various modes - each PRU can operate independently, either PRUs together or PRUs along with the ARM core. Offloading tasks to PRU ensures guaranteed real-time execution with reduced load on primary CPU, the ARM Cortex-A8 processor. The PRUs use shared memory and interrupts for communication between the PRUs and between PRU and ARM core.

PRU helps in building programmable solutions with less external components, at low cost, with higher reliability, and real-time programmability. The Switched Central Resource (SCR) is used by the PRU for low-latency interaction with other resources inside the PRUSS. Open Core Protocol (OCP) is used for accessing resources within the SoC [3].

The versatile nature of PRUSS makes it the most appropriate choice for developing various applications consisting of real-time tasks or subsystems. PRU has been used for various simple to very complex applications like stepper motor control units, sensor interfaces, camera and LCD display interfaces and industrial communication protocols [2].

PRU is very useful for high-speed applications because it can service the hardware with no interruptions due to Linux context switching, and no overhead is experienced by the main ARM processor.

A. *Programming the PRU*

For utilizing the PRU, the following are required.

1. Linux Kernel Driver

   Initially PRUs were using Userspace IO (UIO) driver uio_pruss. Now the implementation is being shifted to remoteproc framework along with rpmsg (remote processor messaging) virtio (virtual IO) devices with pru_rproc [4].

2. Application and Kernel Loaders

   They load the PRU firmware to PRU's memory area and control the PRU execution from the user space. The userspace-PRU control is done by application loader and kernelspace-PRU control is done by the kernel loader [5].

3. Device Tree Entry

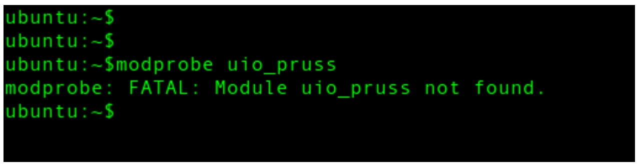
The Device Tree is a data structure for describing hardware [6]. The information is passed to the OS at boot time. It gives information about pin multiplexing according to the mode required, which capes and drivers to use etc. Device tree overlays enable the user to make runtime modifications to disable unwanted pins and to use new hardware peripherals dynamically.

4. Firmware

The firmware is the program executing on the PRU which can be written either in assembly or C. The PRU has small, RISC ISA with approximately 45 instructions which completes in a single cycle allowing 100% predictable timing. PASM is a command line driven assembler for the PRU cores which converts assembly source files to loadable binary data. Currently TI has developed its own PRU assembler which is more flexible than PASM but requires more command line options and a linker command file [7].

B. *Setting up PRU*

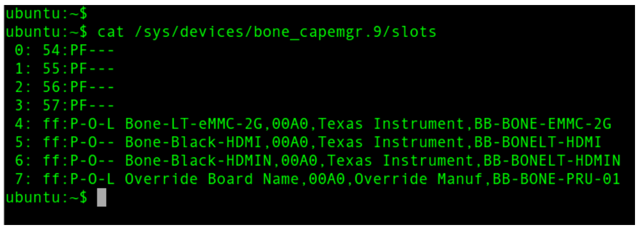
This paper discusses the steps involved in loading uio_pruss module on a Beaglebone Black Rev C. Before using the PRU it must be ensured that the uio_pruss kernel module is loaded.



```
ubuntu:~$
ubuntu:~$
ubuntu:~$modprobe uio_pruss
modprobe: FATAL: Module uio_pruss not found.
ubuntu:~$
```

Fig. 1. Checking with modprobe

The status of uio_pruss kernel module can be known by using the command **modprobe uio_pruss**. If *module not found* error is obtained, then kernel must be recompiled with the uio_pruss module selected.



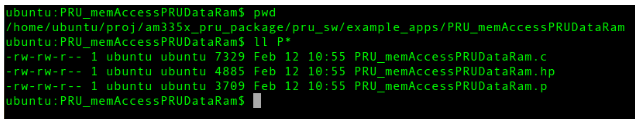
```
ubuntu:~$
ubuntu:~$ cat /sys/devices/bone_capemgr.9/slots
 0: 54:PF---
 1: 55:PF---
 2: 56:PF---
 3: 57:PF---
 4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
 5: ff:P-O-- Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
 6: ff:P-O-- Bone-Black-HDMIN,00A0,Texas Instrument,BB-BONELT-HDMIN
 7: ff:P-O-L Override Board Name,00A0,Override Manuf,BB-BONE-PRU-01
ubuntu:~$
```

Fig. 2. Enabling the PRU through device overlays

The device tree must be configured using device tree compiler to enable the PRUSS and to configure the pins as per requirement. The PASM assembler and drivers for PRU can be cloned from github [8]. After cross-compiling and copying PASM binary and libraries to respective locations, the Beaglebone Black is ready to execute PRUSS programs.

C. *Programming the PRU*

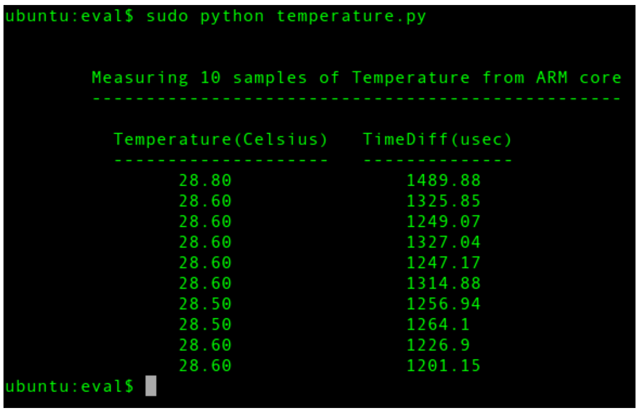
This has two components - assembly/C code for PRU i.e. firmware and C/Python code for the ARM processor i.e. host program.



```
ubuntu:PRU_memAccessPRUDataRam$ pwd
/home/ubuntu/proj/am335x_pru_package/pru_sw/example_apps/PRU_memAccessPRUDataRam
ubuntu:PRU_memAccessPRUDataRam$ ll P*
-rw-rw-r-- 1 ubuntu ubuntu 7329 Feb 12 10:55 PRU_memAccessPRUDataRam.c
-rw-rw-r-- 1 ubuntu ubuntu 4885 Feb 12 10:55 PRU_memAccessPRUDataRam.hp
-rw-rw-r-- 1 ubuntu ubuntu 3709 Feb 12 10:55 PRU_memAccessPRUDataRam.p
ubuntu:PRU_memAccessPRUDataRam$
```

Fig. 3. PRU programming components

The C/Python host program performs the PRU initialization, setting up the interrupts for PRU to notify the host that PRU execution is done, loading the compiled binary to PRU instruction memory and start PRU execution, etc. The PRU assembly/C code performs the desired functionality by manipulating IO pins, sharing the accessed values with host etc. In the same host program, more than one program can be executed on the PRU after each program's execution is completed.

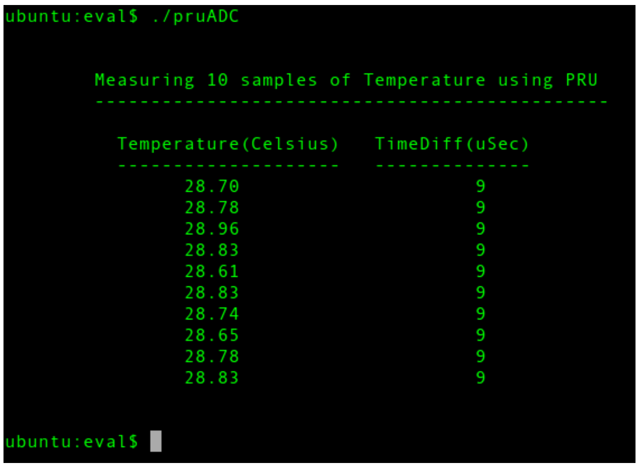


```
ubuntu:eval$ sudo python temperature.py


         Measuring 10 samples of Temperature from ARM core
         ------------------------------------------------------

           Temperature(Celsius)   TimeDiff(usec)
           --------------------   --------------
                  28.80                 1489.88
                  28.60                 1325.85
                  28.60                 1249.07
                  28.60                 1327.04
                  28.60                 1247.17
                  28.60                 1314.88
                  28.50                 1256.94
                  28.50                 1264.1
                  28.60                 1226.9
                  28.60                 1201.15
ubuntu:eval$
```

Fig. 4. Console output showing execution time of program from ARM core



```
ubuntu:eval$ ./pruADC


         Measuring 10 samples of Temperature using PRU
         ----------------------------------------------

           Temperature(Celsius)    TimeDiff(uSec)
           --------------------    --------------
                  28.70                       9
                  28.78                       9
                  28.96                       9
                  28.83                       9
                  28.61                       9
                  28.83                       9
                  28.74                       9
                  28.65                       9
                  28.78                       9
                  28.83                       9


ubuntu:eval$
```

Fig. 5. Console output showing execution time of program using PRU

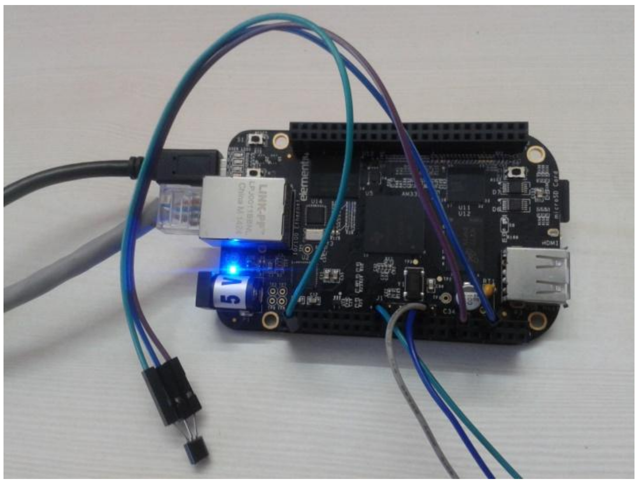


Fig. 6. Experimental Setup

III. Evaluating the PRU

The PRU is analysed by means of real-time acquisition of the ambient temperature of a room using a temperature sensor, LM35.

A. *First Example Program*

For the first example, a BeagleBone Black, running Ubuntu 12.04 armhf, is connected with an LM35 to monitor the room temperature.

A program, running on ARM core, is executed to capture 10 readings from LM35. Using the same setup, another program utilizing ARM core and PRU is executed. From the following plot the difference of speed of operation for collecting 10 temperature readings with and without PRU is easily visible.

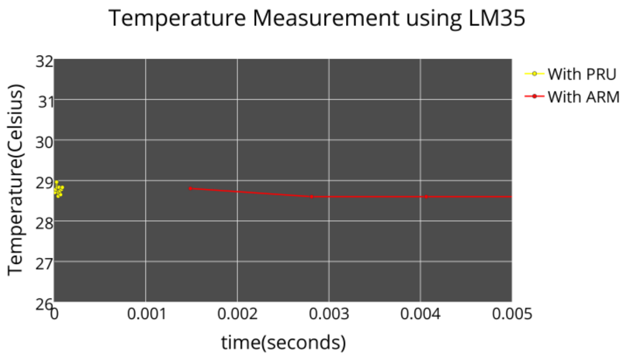
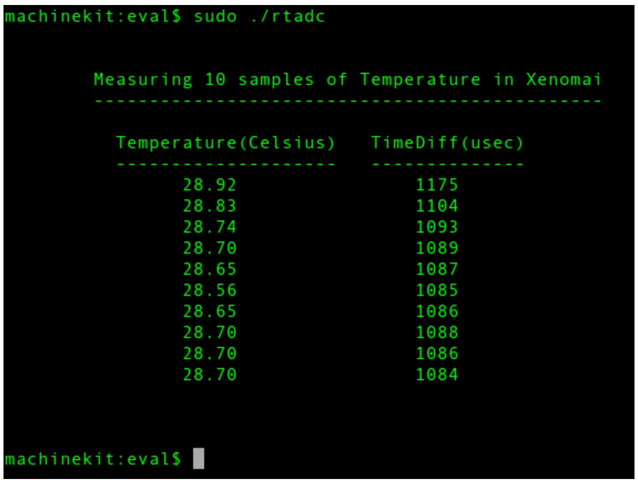


Fig. 7. Plotting the temperature values of both experiments

B. *Second Example Program*

In the next example, a BeagleBone Black running Debian with Xenomai [9], a real-time kernel which runs along with the Linux kernel and can be used for real-time applications, is used. A Xenomai application is used to fetch 10 readings from an LM35 temperature sensor. The same setup is used to execute a program using PRU and without the help of Xenomai.



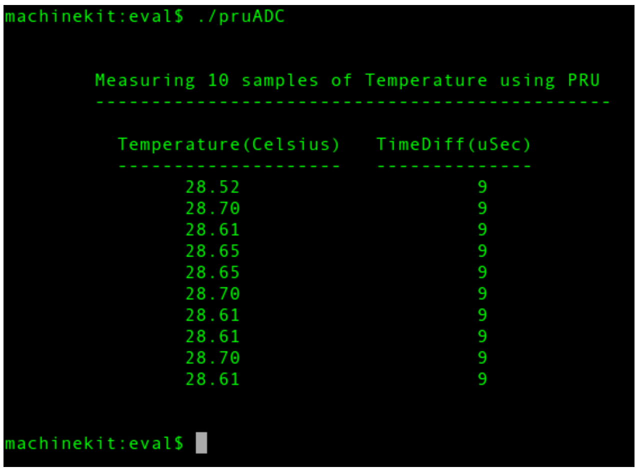
```
machinekit:eval$ sudo ./rtadc


          Measuring 10 samples of Temperature in Xenomai
          ----------------------------------------------

             Temperature(Celsius)   TimeDiff(usec)
             --------------------   --------------
                    28.92                1175
                    28.83                1104
                    28.74                1093
                    28.70                1089
                    28.65                1087
                    28.56                1085
                    28.65                1086
                    28.70                1088
                    28.70                1086
                    28.70                1084


machinekit:eval$ 
```

Fig. 8. Console output showing time taken for each reading using Xenomai



```
machinekit:eval$ ./pruADC


          Measuring 10 samples of Temperature using PRU
          ----------------------------------------------

             Temperature(Celsius)   TimeDiff(uSec)
             --------------------   --------------
                    28.52                     9
                    28.70                     9
                    28.61                     9
                    28.65                     9
                    28.65                     9
                    28.70                     9
                    28.61                     9
                    28.61                     9
                    28.70                     9
                    28.61                     9


machinekit:eval$ 
```

Fig. 9. Console output showing time taken for each reading using PRU

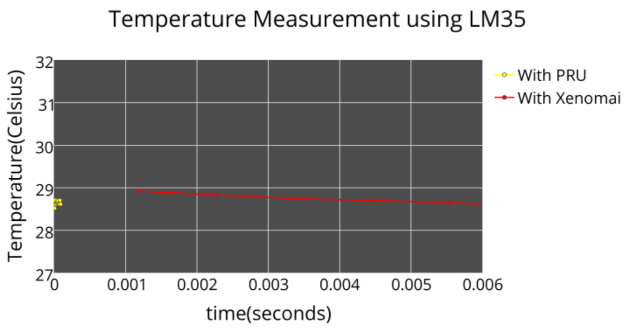


Fig. 10. Plotting the temperature values of both experiments

CONCLUSION

The main purpose of this paper is to evaluate the usability of PRUSS for real-time applications. The PRU helps to extend the capability of the AM335x SoC for performing high-speed data access.

The first example program compares the time difference between successive temperature readings by using ARM core alone and with PRU. The average time difference between successive temperature readings is on an average 1300 microseconds in the case of ARM core and approximately 10 microseconds while using PRU.

The second example program shows the time difference between successive temperature readings using Xenomai alone and using PRU alone. Here also the average time difference between successive temperature readings is on an average 1000 microseconds in the former case and approximately 10 microseconds in the latter.

From both the examples it is evident that PRU performs much faster than ARM core or Xenomai making it more suitable for real-time applications with fast response time.

ACKNOWLEDGMENT

We sincerely acknowledge the immense contribution of Er. Shoukath Cherukat for his guidance, encouragement and supervision during the period of this work.

REFERENCES

[1] Matt Richardson, Getting Started with BeagleBone : Linux-Powered Electronic Projects With Python and JavaScript, 1st ed. O'Reilly, October 2013.
[2] Melissa Watkins, Carlos Betancourt, Ensuring real-time predictability : Leveraging TI's Sitara Processors Programmable Real-Time Unit, TI Whitepaper, July 2014.
[3] http://mythopoeic.org/BBB-PRU/am335xPruReferenceGuide.pdf AM335x PRU-ICSS Reference Guide.
[4] Ron Birkett, Enhancing Real-time Capabilities with the PRU Sitara Boot Camp, Sitara ARM Processors, Oct 2014.
[5] http://processors.wiki.ti.com/index.php/PRU_Linux_Loader PRU Linux Loader.
[6] http://devicetree.org/
[7] Derek Molloy, Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux, 1st ed. Wiley, 2014.
[8] https://github.com/beagleboard/am335x pru package
[9] https://xenomai.org/embedded-hardware/