ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPR003

EXPLORING EMBEDDED SYSTEMS' DEDICATED CORES FOR REAL-TIME APPLICATIONS

P. H. Nallin[†], R. C. Ito, J. G. R. S. Franco, A. R. D. Rodrigues
Brazilian Synchrotron Light Laboratory, Campinas, Brazil

Abstract

Developments and research in high technology leads to powerful and sophisticated machines which are highly important for many scientific fields. Considering real-time applications, however, these systems tend to become non-deterministic and users may find themselves inside a not completely controllable environment. Exploring open-hardware single board computers with a system-on-a-chip which usually runs an operational system on their main processor(s) and also have real-time units is a good alternative. These real-time units are designed as a microcontroller embedded on the chip where a firmware is loaded, runs concomitantly and exchanges data with the main system. As a result, it is possible to achieve performance increase, high temporal resolution and low latency and jitter, features that are widely desired for controls and critical data acquisition systems. This system architecture allows moving real-time data into high level servers, such as Redis (Remote Dictionary Server) and EPICS, easily. This paper introduces and shows uses of Beaglebone Black, an inexpensive single-board computer, its Programmable Real-Time Units (PRUs) and data sharing with Redis data structure.

INTRODUCTION

Designing a new 3 GeV and ultra-low emittance synchrotron machine, brings along many technological challenges. In the case of controls systems, it impacts on controlling and monitoring a large variety of equipment, including the ones that are very accurate and/or critical to light generating. As a solution for both deterministic and general controls, it has been chosen to use an open hardware single board computer, Beaglebone Black [1], which comes with two embedded Programmable Real-Time Units (PRUs). Running on an embedded linux environment, applications have fast hardware access and are integrated to Controls System network. Sirius, the future fourth-generation Brazilian Light Source, comes to the final phase of systems installations for machine engineering. Among them, there are some designed by Controls Group using the Beaglebone Blacks and Programmable Real-Time Units.

USE OF BEAGLEBONE BLACK IN BRAZILIAN LIGHT SOURCES

UVX Facility

Regarding controls system upgrades in UVX facility, the pioneer Brazilian light source, it is remarkable when first CPUs (Z80 and eZ80) were replaced by a commercial fanless single board computer, Advantech PCM-4153F, in 2010, increasing reliability and ease of maintenance.

In the beginning of 2016, the first Beaglebone Black single board computers were introduced to UVX facility controls system, in order to replace older CPU generations [2], either because of electronic components unavailability or outdated equipment. Already intended to be used in Sirius accelerator, it has also been a great bench test for Beaglebone's embedded system.

Since then, running a Debian Linux distribution, it has been detected only one issue concerning hardware/software operation, a possible freezing after a warm reboot, which was corrected in a newer kernel version.

Sirius

Chosen to be the main distributed core in Sirius controls systems, as shown in Fig. 1, there will be more than 350 units running full-time in Sirius applications, such as vacuum system monitoring, power supplies, pulsed power electronics control and temperature acquisition. The choice was made considering its performance, low cost and open hardware project.

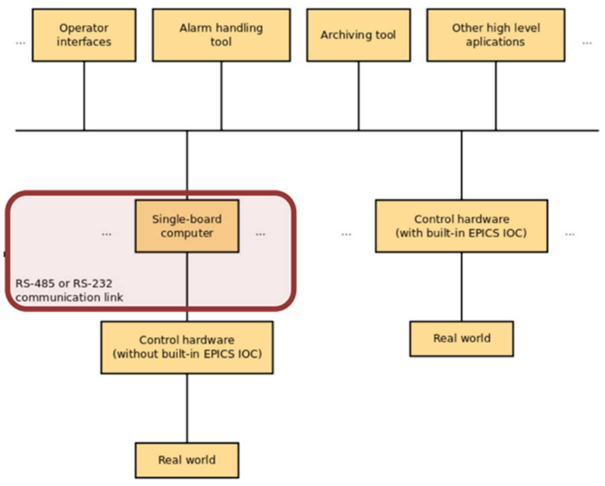
...
Operator interfaces
Alarm handling tool
Archiving tool
Other high level aplications
...
...
Single-board computer
...
...
Control hardware (with built-in EPICS IOC)
...
RS-485 or RS-232 communication link
Control hardware (without built-in EPICS IOC)
Real world
Real world

Figure 1: Sirius Controls System simple layout, considering distributed cores.

However, some of applications have requirements to be performed in real-time systems for several reasons. High amount of data transfer, synchronized operations, low jitter and great time accuracy and determinism.

PROGRAMMABLE REAL TIME UNITS

Beaglebone Black is based on a System on a Chip (SoC) AM335x family, designed and manufactured by Texas Instruments. As subsystems, it has two independent real-time

† patricia.nallin@lnls.br

modules, called PRUs (Programmable Real-Time Units), whose block diagram can be described in Fig. 2.

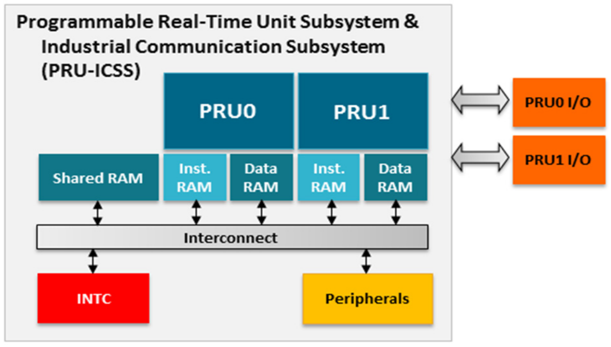

Figure 2: Texas Instruments' PRU subsystem (http://processors.wiki.ti.com/index.php/PRU-ICSS).

Running on a 200 MHz clock, these 32-bit RISC cores are able to implement tasks with elevated real-time constraints. and they have multiple features:

- 8 KB instruction memory
- 8 KB data memory
- 12 KB shared RAM between PRUs and system main core
- Interrupt controller
- Access to some Beaglebone's GPIOs

As an execution model, no pipelining is implemented, which leads to well-known instructions execution time. Multiple addressing modes are available and the reduced instruction set can be divided into three main categories: flow control, arithmetic/logical operation and register data load/store.

Assembly Coding

Despite the effort that must initially be taken, assembly coding leads to a better and deterministic performance of the system. The assembler designed to compile the code is PASM Tool, developed by Texas Instruments, which builds the needed binary file to be loaded into PRUs.

Some available resources increase assembly code development for PRUs and future maintenances. It is possible to have a header file, macro and structure definitions, for example.

C/C++ Coding

Writing source codes in C/C++ language for PRUs is an alternative. A compiler is available through Code Composer Studio (CCS), the Integrated Development Environment provided by Texas Instruments. Even though code writing is faster and optimization options are provided, system responses may be very different compared to assembly coding. The final application and requirements will determine whether C/C++ may be used to achieve desired performance.

Carrier Library

Loading built binary outputs into a PRU and executing an application is also an important step. This is done in userspace level of operating system and consists basically on writing values to specific regions of processor memory.

A C library called libprussdrv [3] can be used as a driver to PRUs, such as mapping memories, writing or reading from it, receive and send interrupts between systems.

GPIO Access

Machine controlling interfaces always require hardware control and access. General purpose I/Os directly accessed by PRUs operate at 3.3 V and have 3.08 ns rise and 2.64 ns fall time.

Additionally, compared to high level userspace GPIO access, controlling these signals through PRU subsystem has a faster response and toggling time can be easily handled by the user.

Applications in Sirius Light Source

Programmable Real-Time units will be used in some controls subsystems where time constraints are tight. Aiming three different tasks, projects developed are described in further sections.

HIGH-PERFORMANCE SERIAL INTERFACE

The first and largest application considering Beaglebone Black's PRU is an interface for fast serial communication, which can reach data rates up to 15 Mbps, named PRUserial485.

Motivation

As a 4th-generation light source, Sirius has been designed to have equipment with great stability and reliability. As modularity is present, it is also important to guarantee that synchronous operations are performed with lowest time difference between systems.

This system has been designed mainly for Sirius' power supplies [4], which are digitally controlled at 6 Mbps. Communication is based on a proprietary and light-weighted protocol, named Basic Small Message Protocol (BSMP), along with a deterministic hardware and software interface.

This leads, for example, to the synchronized execution of current ramps to increase beam energy at booster accelerator.

Hardware Interface

PRU systems already have a serial module (UART). However, configuring high baud rates is limited. For this reason, it has been chosen to use an external UART, MAX3107, controlled through SPI interface.

The baseboard designed for this application is a multi-serial cape, with PRU complementary hardware and other features (standard FTDI, interface SPIxxCON [5] and a hard-reset module). Serial communication physical layer is based on RS-485 standard, where communication chain can reach up to 10 m cable length at 10 Mbps.

PRU external hardware, shown in Fig. 3, is quite simple and composed of the external UART (MAX3107 IC), isolated RS-485 transceiver (IL3685 IC), transmission line fail-safe and termination resistors, DC/DC converter (DCR010505 IC) and an electrical and optical inputs for timing system (synchronized operations).

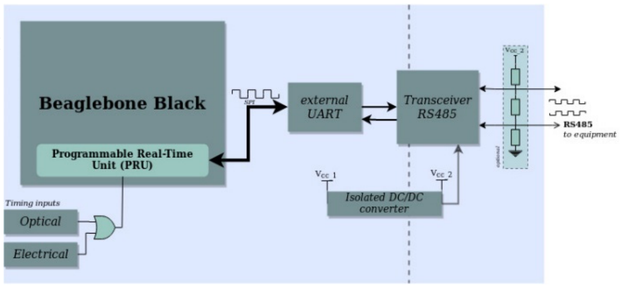

Figure 3: Hardware integration for the high-performance serial controller.

Transmission line termination resistors are manually enabled. The external UART and timing inputs are directly attached to PRU pins, once they will be controlled and used by it.

AM335x Memory Mapping

Data sharing between PRUs and userspace environment is always performed by shared memory access (Fig. 4).

Considering PRUs' shared RAM, some bytes are reserved for device configuring values and data flow control. Yet, 5 KB are reserved for serial sending and 5 KB for serial reading, allowing large amount of data transfer at one time with no delay between bytes.

Larger data arrays can be stored in a reserved region of external on-board DDR memory, defined by blocks 0 to 3, which can be accessed by both main core and PRUs through memory mapping. It was designed especially for ramping power supplies, where points may be continually sent to their controllers, as reference setpoints. Each current value is represented by a floating point, i.e., a 4-byte variable.

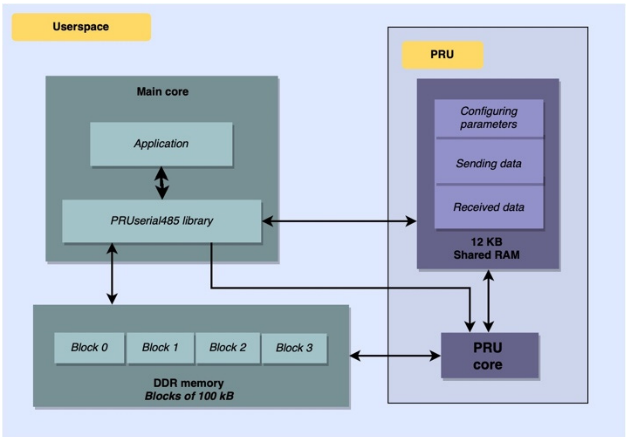

Figure 4: Memory mapping for PRUserial485 application.

Coding

Coding for this application can be divided into three different and linked process:

- PRU assembly firmware
- C library for PRU configuring and memory transfers
- Python user interface

Assembly language has been chosen in order to have high performance and process repeatability. Macros and subroutines were written to ease code development. At starting, external UART is configured and PRU enters in normal operation mode, where idle state is waiting for user data to be sent through serial network and wait for a response from destination. Synchronous operation is the second mode available, making PRU and serial network dedicated to timing input signals.

A C library was developed to load code into PRU and interact with it. There are 15 functions available at the moment, that can be grouped into general purpose (open, close, data write, data read, board address), curves handling (load curve, set/read current curve block, set/read current curve pointer) and sync operation (start, stop, status, read/clear pulse count). This library is compiled and installed into Beaglebone Black.

Getting to higher interfaces, a Python module has also been developed, using ctypes, contemplating all the 15 functions. This module is also installed to operational system.

Synchronous Operation Mode

Sirius power supplies must perform synchronized adjusts during booster ramping, orbit correction, magnets cycling and migration mode. For booster energy ramping, for example, curves with 3920 points must be run at 2 Hz. For that purpose, a PRU serial mode was implemented to deliver serial messages with low jitter to power supplies' controllers. Concerning serial network, there are two main sync modes: after a triggering a timing pulse, PRU interface sends through RS-485:

1. Broadcast command
2. Command addressed to a specific device

When sync mode is enabled, PRU is completely dedicated to receiving timing signals, polling a GPIO.

In the first model, after receiving a timing pulse, serial interface sends a broadcast message to all devices in the line. Final action depends on how the power supply is configured: it could be a setpoint implementation or start of a magnet cycling, for example.

The second one sends a sequence of setpoints each time a timing pulse is received, following ramp curves that user has loaded previously. Ramp curves are stored in DDR memory and they are subdivided into four different groups. A group is composed of four curves of up to 6250 points (4-byte floating points) each.

In this case, it is possible to changes curves on-the-fly by loading them into a different block as well as changing pointer for next point of curve in execution. Also, two more parameters should be configured: execute curve once or multiple times and intercalate normal write/read messages between timing commands or only after the last curve point.

The two modes have different latency, smaller than two bytes-time yet. However, message jitter after a timing signal is quite reasonable. Values are shown in Table 1. After sending a sync serial message, the interface requires a recovery time of 3.1 µs before polling the next pulse.

Table 1: Characterization of Different Serial Message Types that are Sent after Detecting a Sync Trigger

PRU sync command	Message size [bytes]	Latency [µs]	Jitter [ns]
Broadcast	6	1.011	13.44
Single setpoint adjust	10	1.438	52.73
Four setpoints adjust	22	2.188	82.43

As expected, the case of a broadcast command shows a lower jitter once data is stored directly in shared RAM instead of DDR memory, which is external to the SoC.

Current Status and Next Steps

Several tests were made with this high-performance serial communication and reliability has been demonstrated to be elevated. The possibility of changing curves on-the-fly is a flexibility that allow accelerator physicists improve mainly machine commissioning.

As an upgrade, it has been planned to use the second PRU unit to increase reliability concerning eventual lost timing pulses.

MULTI-PURPOSE COUNTING SYSTEM

A general-purpose counting system is very modular and can be used in many applications concerning accelerators or beamlines.

Motivation

Some diagnostics devices in Sirius accelerators present a readable signal output as pulse trains. Counting the amount of pulses received in a time-based period gives some quantitative or qualitative results, depending on the application.

By the moment, two devices have been used and they respond with pulses: Beam Loss Monitors (BLM - differential model), from Bergoz, and in-house developed gamma detectors. Maximum pulse rate is 10 MHz for BLMs and 1 MHz for gamma detectors.

Hardware Interface

Being the first Controls Group project to be built with Power over Ethernet (PoE) interface, hardware interface is easily powered and accessed with only one network cable. Also, it has an analog-to-digital driver for differential Bergoz interface as well as power levels required for both devices. The baseboard has been designed for controlling two Bergoz units and six general-purpose digital HCT channels, which are used with Sirius Gamma Detector (Fig. 5).

Digital hardware is based on digital latches that can detect fast rising edges. Their output and reset signals are directly connected to PRU GPIOs.

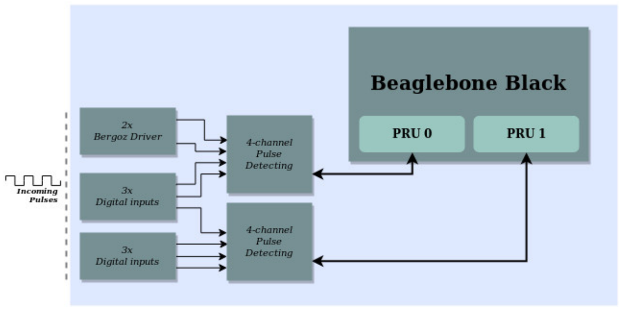

Figure 5: Hardware interface for general-purpose counting system.

Coding and Memory Mapping

Compared to the high-performance serial communication, this application is simpler and do not have a very restricted time constraint. For starting, stopping counters and share final 32-bit count results with Linux users, PRU shared RAM is used. Developing PRU applications leads to the same linked process:

- PRU assembly firmware
- C library for PRU configuring and memory transfers
- Python user interface

PRU firmware was first written in C, achieving maximum count rate at 4 MHz for one active channel. An assembly version was also written to compare performance and it is faster (table 2). Having a total of eight independent channels, each programmable unit controls four of them.

The C library has three implemented functions: open/close PRU application and counting for a time period. This is also mapped to Python using ctypes and both language libraries are installed to Beaglebone's operational system.

Maximum counting rates depend on the number of active channels per PRU.

Table 2: Maximum Counting Rates

Active channels	Max count rate [MHz]
1	14.29
2	12.50
3	11.11
4	10.00

Current Status and Next Steps

PRU-based counting system had UVX facility as its bench test for BLMs and eight units have already been used in initial beam tests in Sirius booster.

Concerning gamma detectors and counting system, they have been under tests with successful results and will be permanently installed soon in Sirius storage ring, adding around 60 Beaglebone Blacks to controls system network.

WATER LEAK DETECTOR

This project, which is currently in final development phase, has the goal of detecting water leaks along some structures of Sirius accelerators.

Motivations and Hardware Interface

When Sirius is under operation, the tunnel is inaccessible and thus it will be no longer possible to monitor hydraulic connections and hoses. Yet, some small leaks may not be detected easily. For that purpose, a water leak detector has been designed and is based on the principle of signal reflexion on a coaxial mismatched transmission line, caused by a few millilitres of water absorbed by a capacitive sensor.

Initially, all Sirius' girders will be monitored, where offset from one another is 5 m. The baseboard includes a Beaglebone Black and an analog conditioning electronics circuitry. Triggering pulse is controlled by PRU and reflected one is attached to another PRU GPIO (Fig. 6).

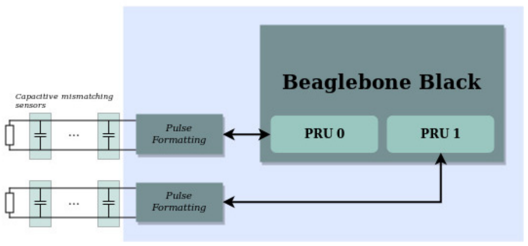

Figure 6: Hardware interface for general-purpose counting system.

Distance measurements, concerning signal velocity of propagation through a coaxial cable, have maximum errors of 1.22 m, which means 0.61 m one-way maximum error distance.

Future Developments

It is intended to manufacture all capacitive sensors in next months and integrate this application to controls system as soon as possible, once water cooling installations are almost finished.

EPICS AND REDIS INTERFACE

A very interesting feature of using Beaglebone's PRUs is that user is already inside an embedded linux. Having the possibility of developing real-time applications in such environment makes system integration easier, cleaner and time saving.

Once PRUs may be controlled either through a Python module or a C library, from the same SoC userspace, writing upper-layer codes to interact with them and make information available to other applications might reduce system complexity. Two large important and open source tools are currently in use in order to share obtained data from Sirius subsystems.

EPICS

Sirius controls system is EPICS based and its single board computers are also configured to fully integrate the network.

The minimal system image stored in Beaglebone Blacks has the latest stable release EPICS base installed (R3.15) as well as some other modules, such as asynDriver and StreamDevice. As consequence, it is possible to run EPICS IOCs (input/output controllers) directly from each device.

Redis

Redis database, an open source in-memory data structure store [6], is a tool that has been largely used and in use in a few applications in LNLS campus since 2014.

Its main application has been demonstrated to be robust and reliable while running on a Beaglebone Black, which monitors and controls the campus power generator.

Similarly to EPICS base, Redis server can be easily installed in the embedded system and interfaced with a Python module, getting closer to hardware access. The minimal controls system's embedded image also provides a Redis server installed and enabled, with the goal of improving current applications. For the moment, all Beaglebone Blacks have their functionality information stored in a Redis database. The Counting System also stores its counter values in their embedded Redis server.

CONCLUSION

Even though some applications are not completely running in-situ yet, Beaglebone Black has been demonstrated to be a reliable, powerful and cheap solution as a controls system node.

Having both embedded Linux and a dedicated core in the same SoC reduces costs and system overall complexity, allowing developers to design time-critical applications with the advantage of sharing data with largely used tools, such as EPICS and Redis.

Shown applications have great performances, mainly the one designed to interface power supplies, the high-performance serial interface, which has already passed exhaustive tests and is currently used for booster and subsystems commissioning. Future improvements are possible and some are already planned.

REFERENCES

[1] BeagleBone, `http://beagleboard.org/bone`

[2] S. Lescano, E. P. Coelho, G. R. S. Franco, P. H. Nallin, G. C. Pinton, and A. R. D. Rodrigues, "UVX Control System: An Approach with Beaglebone Black", in *Proc. PCaPAC'16*, Campinas, Brazil, Oct. 2016, pp. 91-93. `doi:10.18429/JACoW-PCAPAC2016-THPOPRPO03`

[3] AM335x PRU PACKAGE, `http://github.com/beagleboard/am335x_pru_package`

[4] C. Rodrigues *et al.*, "Overview of Sirius Power Supply System", in *Proc. 10th Int. Particle Accelerator Conf. (IPAC'19)*, Melbourne, Australia, May 2019, pp. 1230-1232. `doi:10.18429/JACoW-IPAC2019-TUPMP002`

[5] G. R. S. Franco *et al.*, "Software and Hardware Design for Controls Infrastructure at Sirius Light Source", presented at the 17th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'19), New York, NY, USA, Oct. 2019, paper MOPHA031, this conference.

[6] Redis, `https://redis.io/`