SOFTWARE ARCHITECTURE FOR NEXT GENERATION BEAM POSITION MONITORS AT FERMILAB

J. S. Diamond[#], Fermilab, Batavia, USA

Abstract

The Fermilab Accelerator Division / Instrumentation Department develops Beam Position Monitor (BPM) systems in-house to support its sprawling accelerator complex. Two new BPM systems have been deployed over the last two years – one upgrade and one new. These systems are based on a combination of VME and Gigabit Ethernet connected hardware and a common Linux-based embedded software platform with modular components. The architecture of this software platform and the considerations for adapting to future machines or upgrade projects will be described.

INTRODUCTION

The Fermilab Booster is a synchrotron accelerator with a circumference of 474 meters which accepts 400 MeV protons from the Linac, accelerates to 8 GeV in less than 67 milliseconds to be injected into the Recycler. In 2018 the Booster BPM data acquisition system was upgraded to a VME-based system based on in-house developed digitizer and timing modules. At the same time the design of a VME-based BPM system for Fermilab's newest accelerator, the Integrable Optics Test Accelerator (IOTA) was being developed. The decision was made to re-use as much hardware, firmware and software as possible from the Booster BPM upgrade project.

HARDWARE & FIRMWARE

Both the Booster BPM upgrade project and the IOTA BPM project are based on VME 64x crates supplied by Weiner. In the case of the Booster, six crates are utilized to instrument all the BPMs while IOTA uses only one. Each crate contains a Single Board Computer (SBC), a Timing and signal distribution board and multiple digitizer modules. The Booster BPM system uses an Artesyn MVME-8100 SBC with a QorIQ processor and 2GB of system RAM. The IOTA BPM system uses a Concurrent Technologies 405x SBC with an Intel Core Duo processor and 2GB of system RAM.

The Timing module is based on a design originally developed for the Main Injector and Tevatron BPM systems. The timing module decodes the Fermilab site-wide machine clock, TCLK and synchronizes with the machine RF.

Each digitizer module receives ADC clock and trigger signal from the timing module. The raw signals from the BPM plates are passed through an analog transition module and connected to an ADC channel on the digitizer module. The digitizer modules run at 250 MSPS and filter the data through a down converter and into on-board RAM.

In the Booster BPM the analog transition modules are present in the VME crate and use the VME bus for power. The data acquisition software can detect these modules and verify that they are present but otherwise does not interact with them. In the IOTA BPM these modules are external to the VME crate and powered by their own NIM crate and have no interaction with the data acquisition software. Plans for a future BPM system are incorporating a Raspberry Pi-based controller in the NIM crate for controlling attenuation and gain settings on the analog transition modules.

EMBEDDED LINUX STACK

Buildroot is an open source embedded Linux build system that automates the construction of a cross-compile toolchain, Linux kernel and root filesystem. Building a Linux kernel and root filesystem from scratch gives the developer control over the cross-compile toolchain, which support software is present and which kernel options are used. Using Buildroot also allows us to achieve a system footprint of less than 30 MB.

DEVICE DRIVERS

Communication with hardware devices over the VME bus was achieved using the mainline VME driver introduced into the Linux kernel in version 3.10. This driver allows developers to interact with VME attached device from within a Linux kernel module (LKM) much like a PCI or USB device. The mainline VME driver supports both the Universe II and TSI-148 VME bridge chips through a common API.

Each in-house developed hardware device requires an LKM to be developed to facilitate communication between the data acquisition software and the device. In the case of the timing modules this LKM only communicates with one device but the LKM for the digitizer and analog transition modules must support communication with multiple devices. Upon insertion into the kernel the LKM requests access to the VME bus by acquiring a resource from the kernel VME driver. The VME resource is used to scan the VME bus and probe for hardware. Once a hardware device is successfully probed it is registered with the Linux device model as a VME bus device. If the LKM is removed, user space is notified, and the device is removed from the device model as a part of the LKM shutdown procedure.

Access to hardware device registers is provided to user space using the Linux sysfs filesystem. Each register on the hardware is made available as a sysfs attribute that can be read/written through the device's entry in sysfs. This is a useful tool for developers to interact with and

* This work was supported by the DOE contract No. DEAC02-07CH11359 to the Fermi Research Alliance LLC.
# jdiamond@fnal.gov

diagnose hardware directly from the command line using the familiar Linux tools such as 'cat' and 'echo'. Access to the digitizer's memory buffers is provided through the traditional Linux character device interface.

The timing modules and digitizer modules both deliver interrupts to the system to synchronize the data acquisition software with external events. Because the data acquisition software runs in user space and the interrupt handlers exist in kernel space a mechanism for delivering interrupt notifications must be chosen. Generic Netlink is a Linux kernel feature design for passing asynchronous messages to and from the kernel with an API similar to the sockets interface. The user space application creates a socket and binds it to a named socket on the kernel side that will broadcast messages from the interrupt handler. The application that enters into a message handling loop and goes to sleep until a message arrives. In addition to interrupts the LKM developed for each type of hardware device use Generic Netlink to notify user space when hardware is probed and when it is removed from the system.

The development of LKMs for multiple hardware devices with similar hardware and software interfaces resulted in the production of a lot of boiler plate code. To assist in the development of these LKMs and the C++ API to interface with them, a code generation tool named 'drvgen' was developed. The drvgen tool takes in a hardware device interface specification in the form of a comma-separated spreadsheet file. The specification includes the register map, device memory region, interrupt handlers and command functions. The code generated includes the LKM and a C++ library that wraps the sysfs, character device and Generic Netlink interfaces. Modules with stubbed-out functions are generated on the first drvgen run for the developers to add custom code that won't be overwritten by drvgen when the specification changes.

To communicate with the hardware a user space application asks the generated C++ library for a device object or a pool of device objects that represent the hardware detected on the VME bus. The C++ library uses the sysfs, character device or Generic Netlink interfaces to communicate with the LKM and the LKM uses the kernel's VME API to communicate with the hardware. Stubs for interrupt handlers are provided in a separate module for developers to write interrupt handlers. When a feature or register is removed from the specification a kernel message is written to warn the developer that a deprecated feature was used if the application code continues to call it.

Development is underway right now to expand the data throughput to the digitizer modules by utilizing their front panel Gigabit ethernet interface through a rack-mount ethernet switch. Both the Booster BPM and IOTA BPM systems will be upgraded once these interfaces are established.

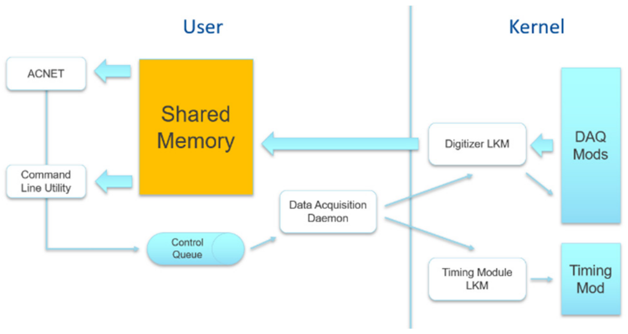

Figure 1: BPM Data Acquisition Software Process Model.

DATA ACQUISITION SOFTWARE

The Beam Position Monitor data acquisition software is responsible for arming the digitizer modules ahead of a measurement and collecting data for each measurement after the digitizer modules have acquired it. Measurement data must be collected before the next machine cycle begins so the digitizer modules can clear their buffers in anticipation of the next measurement.

The data acquisition software runs as a Linux daemon process (see Fig. 1) and is started as a part of the system boot process. Threads are spawned to handle requests from external interfaces over a message queue, coordinate machine state through clock events decoded by the timing module and to collect data from the digitizer modules into a shared memory region. The daemon process writes log messages to the Fermilab-developed TRACE facility to assist with diagnostics.

The daemon process reads its configuration from a human-readable text file and parses it with the libconfig library. The configuration contains the BPM database, measurement profiles and machine state configuration and various other system level parameters. The configuration data is then made available via a global configuration object and published in the shared memory region to be read by clients. The daemon process can be requested to reload the configuration at runtime via a request sent over the command message queue should changes be made by experts during operation.

Each accelerator at Fermilab has its own state and is synchronized with the rest of the complex using a clock system known as TCLK. The timing module is responsible for decoding this clock signal and generating interrupts when machine events occur. Taking these events and using them to coordinate the position measurements is the responsibility of a State Machine implementation. A custom State Machine implementation was developed for both the Booster and IOTA BPM systems and is specified as a part of the data acquisition software's configuration. The State Machine implementation configures the timing module and responds to the events it generates by configuring the data acquisition system to make position measurements.

The Data Acquisition system is responsible for managing the pool of digitizer modules. The Data Acquisition system operates in 5 different states: Initialization, Ready, Armed, Triggered and Readout. When the daemon process starts up the digitizer modules are initialized according to

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA034

the configuration file and the system transitions to the Ready state while awaiting direction from the State Machine implementation or the command line utility. To prepare for a measurement a message is sent to the daemon process to arm causing the Data Acquisition system to prepare the digitizer modules and transition to the Armed state. From the Armed state the Data Acquisition system can transition to Triggered when the digitizer modules are triggered or back to Ready if the measurement is aborted. Once in the Triggered state the data can be readout into the shared memory region, or it can be armed again. If a readout is requested, then the system enters the Readout state while the data is transferred. If the system is armed again without a readout the data is discarded.

USER INTERFACES

To interact with the data acquisition software an API library was developed that wraps the shared memory and message queue interfaces in C++ classes.

A command line utility was developed to assist with diagnosing the data acquisition software from the console or over the secure shell. All requests that can be processed by the data acquisition software can be sent by an expert using the command line utility. In addition, configuration parameters and measurement data can be inspected from the console using the command line utility. The output of the command line utility can be redirected to a file if the user would like to dump data for offline analysis.

The control system interface to the beam position monitors is different for each machine and as such it was appropriate to develop separate interfaces for the Booster and IOTA or future BPM systems. The control system software framework is called Acsys/FE and runs as a separate process and like the command line utility, uses the library API to read measurement data out of the shared memory region and send requests over the message queue. Future accelerators at Fermilab may use EPICS as the control system and this model will allow us to develop an EPICS interface in place of the Acsys/FE framework.

CONCLUSION

The development of an embedded-Linux based beam position monitor system represents a two-year effort by software developers and electrical engineers in the Accelerator Division, Instrumentation department. The system that has evolved is flexible enough to be deployed in two different accelerators, the Fermilab Booster and IOTA. Plans for a future BPM system at the PIP-2 Integrated Test stand are already in development for 2020 and include a shift from the legacy VME bus to ethernet-attached hardware.