Control System for Fast Components of Electron Beam Welding Machines

Small Experimental Electron Beam Welding Machine at BINP

A research of electron beam technologies is conducted nowadays at BINP. Several electron beam machines have been constructed here including Small Experimental Electron Beam Machine which incorporates some interesting solutions like a magnet and where experiments like 3D-printing with welds have been carried out.

The machine consists of so-called slow and fast subsystems. Slow subsystems successfully handled by CX control system with frequency about 1 Hz. Fast subsystems of the machine like beam deflectors or mechanical moving plates require to be controlled with frequencies from 100 Hz to 2 kHz.

Also the machine frequently undergoes reconstruction and improvements so its control system should be easily updated too.

We decided to implement this fast control system using Raspberry Pi.

Raspberry Pi

Raspberry Pi is a fully featured single board computer with GNU/Linux support.

Version 3 is based on a BCM2837 SoC with 64-bit 1.2 GHz quad-core ARM Cortex-A7 and has 1GB LPDDR2 RAM on board.

It includes following interfaces:
- Ethernet
- 4 x USB
- WiFi and Bluetooth
- 4 x GPIO
- 4 x SPI
- 2 x UART
- HDMI
- SD/MMC

Raspberry Pi also has a wide range of external adapters for Raspberry Pi to be put on top of GPIO.

Raspberry Pi as the core of the control system

The control system for fast components of Small Experimental Electron Beam Machine was called “RPS-CNC.” Its software part was designed to be modular and easily extensible. It has layered structure with increasing abstraction level and specific interface of each layer.

The hierarchy of software part of RPS-CNC is the following:
- Web client with graphical user interface written in JavaScript communicating with the server via WebSocket.
- Python web-server written in asynchronous manner using Asynchronous Library. The server interacts with underlying Python binding of RPi-CNC library.
- Python binding wrapping the C API of the library.
- RPi-CNC library high-level part. It performs task management, receives signals from underlying low-level part and prepares command sequence for device control.
- RPi-CNC library low-level part. It is written in C with optional modules written in Rust. It generates waveforms and signals and receives signals from sensors. It interacts with the Raspberry Pi SoC hardware via gpio library.
- Pigpio third-party library. It provides rich Raspberry Pi management capabilities including tricky way of playing waveforms in real-time with frequency up to 10 kHz using DMA and PWM controllers.

The hardware part includes CAN adapter for Raspberry Pi, stepper motor driver, custom proximity sensors adapter and power supplies for all electronics and stepper motors. All the hardware was assembled into standard 19-inch rack mount chassis and integrated in experimental electron beam machine infrastructure.

RPS-CNC server:
- https://github.com/binp-automation/epics-hot
- https://github.com/binp-automation/epics-hot

EPiCS on Embedded Electronics

Next-gen power supply controllers is another part of our work. Among other functionality they should be integrated into EPICS control system being used at BINP accelerators. Because EPICS could be built for ARM architecture and i.MX7 is equipped with relatively high performance Cortex-A7 core it is reasonable to consider the possibility of running EPICS device support directly on the controller without using any intermediate gateways between the device and control system.

We have made a simple stress test for EPICS device support. The setup was the following: a simple device support program was ran on a VAR-SOM-MX7 and multiple clients were connected to the server via 1 Gbps Ethernet. The clients physically were ran on a single machine but used different MAC and IP addresses. One client wrote a new waveform containing 20000 points of DOUBLE type into PV and the server broadcast these data to other subscribed clients, and this operation was repeated continuously. The delay between the sending of the waveform and its receiving by the last client was measured along with Cortex-A7 CPU load.

There were run two attempts: for 25 and 256 connected clients. For 25 clients this number is usual number of clients connected to such kind of device on VEPP-4, the delay was equal to about 150 ms and the CPU load was 40-60% (the full load of dual-core CPU is 200%). For 256 clients the delay was about 1 second and the load was 80-120%.

Testing program could be found here: https://github.com/binp-automation/epics-stress-test

Rust Programming Language for Control Systems

Rust is a modern programming language. It is declared to be fast and memory-efficient, because it is compiled to native code and has no runtime or garbage collector. Rust is designed to be safe - it guarantees memory and thread-safety at compile-time. Also Rust is actively developed and has a wide range of tools.

Because of these advantages Rust is a good candidate to use in physics control systems development which require stability and performance. Also because of its resource efficiency it is suitable to use Rust in embedded devices and moreover this application is one of the main efforts of Rust development team.

For now we use Rust in device support development for EPICS control system. The advantages of using Rust in such case is that it helps to develop more reliable software and reduces time spent for debugging. We have created EPICS bindings for Rust and implemented the framework for writing EPICS device supports and drivers in Rust.

You may learn more about Rust on its website: https://www.rust-lang.org/

EPICS to Rust bindings: https://github.com/binp-automation/epics-rs
EPICS device support template in Rust: https://github.com/binp-automation/devsup-template
Keyshot 5GHz frequency counter device support using this template: https://github.com/binp-automation/ksfc-devsup