ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WESH3002

CONTROL SYSTEM FOR FAST COMPONENTS OF ELECTRON BEAM WELDING MACHINES

A. V. Gerasev, P. B. Cheblakov, Budker Institute of Nuclear Physics, Novosibirsk, Russia

Abstract

Modern electron beam machines for different applications including welding, additive technologies and etc. consist of many different subsystems, which should be controlled and monitored. They could be divided by so-called fast and slow subsystems. Slow subsystems allow reaction time to be around a couple of seconds that can be implemented using PC. Fast subsystems require time to be around hundreds of microseconds combined with flexible logic.

We present an implementation of such fast system for mechanical moving platform and electron beam control. The core of this system is a single board computer Raspberry Pi. We employed a technique of fast waveform generation using Raspberry Pi on-chip DMA to manipulate stepper motors. Raspberry Pi was equipped by external CAN controller to operate an electron beam via CAN DACs. Special software was developed including libraries for low- and high-level technical process control written in C and Rust; and in-browser graphical user interface over HTTP and WebSockets. Finally, we assembled our hardware inside standard 19-inch rack mount chassis and integrated our system inside experimental electron beam machine infrastructure.

RASPBERRY PI AND ELECTRON BEAM WELDING MACHINE CONTROL

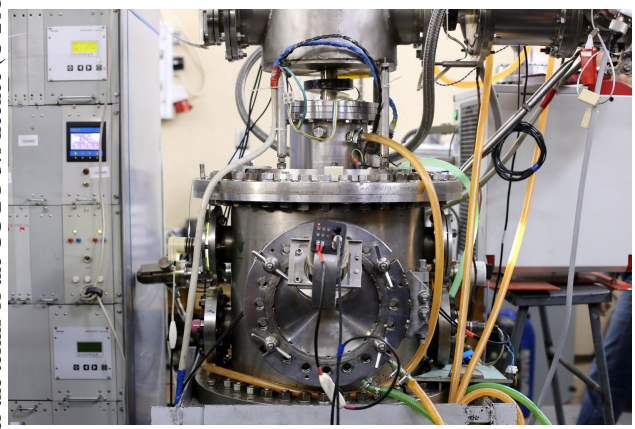

Figure 1: Small experimental electron beam machine in Budker Institute of Nuclear Physics.

Electron beam technologies become increasingly applicable in different areas. Their principle is based on beam of electrons hitting the object in vacuum and locally heating it. This approach is used for precise cutting and welding, and also for 3D-printing. In our institute research of such technologies is carrying out and a couple of different electron beam machines have already been constructed. One of them is experimental small electron beam machine shown in Fig. 1. One of successful experiments on this machine was 3D-printing with wolfram [1].

The machine contains so-called fast and slow components. Slow components like power supply and vacuum subsystem require the reaction time to be about one second and are successfully handled by general-purpose CX [2] control system. But for some experiments it was necessary to handle specific components like beam parameters control subsystem in more fast and precise manner. The components that require such control are called fast components.

Due to the experimental nature of the machine along with providing required performance and precision the control system of fast components should be flexible and easy to develop. We found that Raspberry Pi is the most suitable platform for such system implementation.

Raspberry Pi

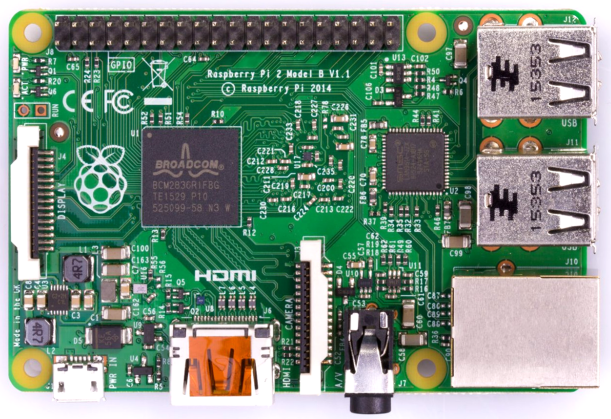

Figure 2: Raspberry Pi 3.

Raspberry Pi is a fully featured single board computer (shown in Fig. 2. There are several versions of this computer. We used versions 2 and 3. Version 2 is based on Broadcom BCM2836 system on chip (SoC) which contains 32-bit quad-core 900 MHz Cortex-A7. Version 3 is based on Broadcom BCM2837 SoC with 64-bit quad-core 1.2 GHz Cortex-A53. There is also Graphics Processing Unit (GPU) on the chip - VideoCore IV 250 MHz and 400 MHz accordingly. Along with CPU and GPU SoC also contains different peripherals devices (almost the same for both chips) including timers, interrupt, direct memory access (DMA), pulse width modulation (PWM) controllers. These models of Raspberry Pi has a lot of external interfaces like Ethernet, 4xUSB, General Purpose Input-Output (GPIO), 4xSPI and 2xUART.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WESH3002

Control System for Fast Components

There are two fast components required to be controlled with fast control system:

- 3 digital-to-analog converters (DAC) setting beam current and x- and y-deflection. The frequency of these DACs is 100 Hz. The communication with them occurs via CAN bus.
- Mechanical moving plate driven by 3 stepper motors controlled via low-level step-dir protocol with the frequency up to 2 kHz.

The first component is effectively handled by PICAN2 Duo Iso adapter mounted on top of the Raspberry Pi (shown in Fig. 3). This adapter is put onto GPIO pins and is controlled by Linux via standard driver that allows user space access via Socket-CAN interface. The latency of non-real-time Linux on Raspberry Pi is small enough to provide 100 Hz communication via CAN bus.

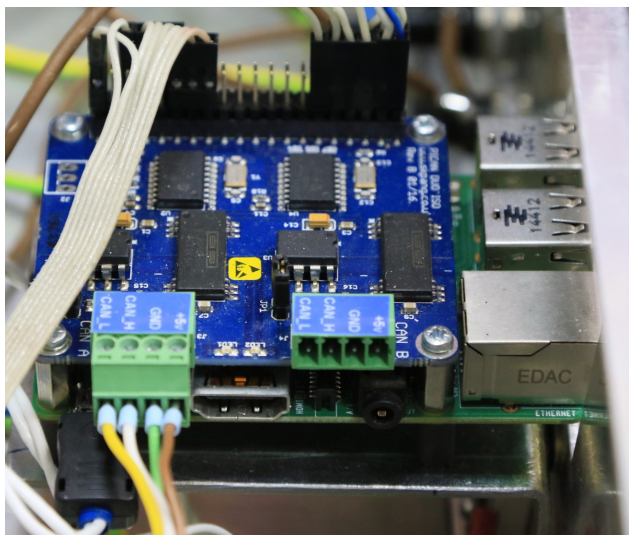

Figure 3: PiCAN2 Duo Iso adapter mounted on top of Raspberry Pi.

To operate the second component the latency of non-real-time Linux is too high. But we have found a relatively tricky way of generating such impulses via Raspberry Pi hardware peripherals - particularly DMA and PWM [3].

DMA on Raspberry Pi is the chain of control blocks where previous block points to the next. The block does very simple operation - it copies specified amount of bytes from specified source address into destination one. All peripherals including GPIO are mapped on the memory so DMA can also control the hardware. PWM controller is used to provide precise delay mechanism. Using this technique Raspberry Pi can generate impulses with the frequency up to 10 kHz and the timing precision is around 1 us that fully meets our requirements.

The controls system for these components were implemented in a modular manner. The stepper motor control was implemented in our librpicnc [4] library that uses pigpio [5] library for hardware access. Our library consists of low- and high-level parts.

- Low-level part receives low-level commands for separate axes, synchronizes them and generates waveforms for pigpio library. This part is written in C and was partially rewritten in Rust [6].
- High-level part is a task manager that receives high-level tasks and translates them to low-level axis commands.

This library provides a C application programming interface (API) to control the moving plate. Also a Python3 binding to this library was implemented. To control moving plate using graphical interface the control server was developed. Control server is aiohttp web server that communicates with in-browser graphical user interface (GUI) via HTTP and Websocket protocols.

VARISCITE VAR-SOM-MX7 SYSTEM ON MODULE

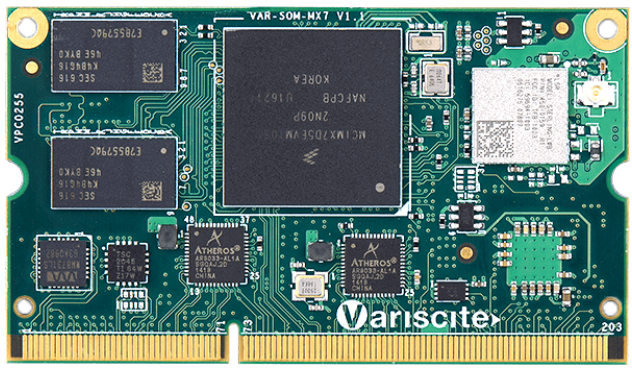

Figure 4: VAR-SOM-MX7 Module.

The fundamental flaw in using Raspberry Pi to control stepper motors in real-time is that its DMA logic is very primitive. It can produce only pre-built fixed sequence of signals in real-time but any branching or feedback handling require non-real-time Linux to be involved.

We were looking for embedded device which is more flexible but still provides real-time operation. We have found Variscite VAR-SOM-MX7 system on module (SoM) (shown in Fig. 4) which is based on NXP/Freescale i.MX7 SoC. i.MX7 is asymmetric multiprocessor chip - it has both dual ARM Cortex-A7 1.2 GHz for running general purpose OS and performing high-level non-real-time tasks and ARM Cortex-M4 200MHz for low-level real-time tasks. i.MX7 Cortex-A7 cores supports GNU/Linux with Debian and Yocto Linux distributives supplied by Variscite, and Cortex-M4 which is usually employed in microcontroller devices supports FreeRTOS.

EPICS DEVICE SUPPORT ON EMBEDDED ELECTRONICS

Our work previously been based on electron beam welding machine control has gradually evolved to VEPP-4 particle accelerator. We are planning to use Variscite VAR-SOM-MX7 as a digital part of power supply controller.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WESH3002

The main VEPP-4 control system is EPICS so it is required for the controller to interact with it. Because EPICS could be built for ARM architecture and i.MX7 is equipped with relatively high-performance Cortex-A7 cores 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 [7] for EPICS device support. The setup was the following: a simple device support program [8] 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 point of DOUBLE type into process variable (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 ran two attempts: for 20 and 256 connected clients. For 20 clients (this number is usual number of clients connected to such kind of device on VEPP-4) the delay was equal to about 100 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%.

So we can make a conclusion that i.MX7 is able to manage a regular load of power supply device support as well as 10 times higher load.

RUST PROGRAMMING LANGUAGE

Rust [6] 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 reasonable to use Rust in embedded devices and moreover this application is one of the main efforts of Rust development team [9].

We decided to try Rust in our tasks. At first we successfully applied Rust for low-level interaction with peripherals on Raspberry Pi.

For now we use Rust in device support development for EPICS control system. The advantages of using Rust in such case is that it simplifies the software development and reduces time spent for debugging. We have created EPICS binding for Rust [10] and implemented device support and driver for Keysight 53220A frequency counter via LXI interface fully in Rust for use on VEPP-4 facility [11]. The template of device support in Rust [12] was created to make implementing device support for other devices easier.

REFERENCES

[1] Yu. I. Semenov *et al.*, "3D printer mockup for manufacturing metal structures from refractory metals using electron beam additive technologies", VI Conference of Lasers and Plasma Technologies (CLAPT–2015), March 24-27, 2015, Novosibirsk, Russia.

[2] D. Bolkhovityanov, P. Cheblakov, F. Emanov, "CXv4, a Modular Control System", in *Proc. 15th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2015)*, Melbourne, Australia, 17-23 Oct 2015. doi:10.18429/JACoW-ICALEPCS2015-WEPGF093

[3] Waveforms using Raspberry Pi DMA, https://hackaday.io/project/158810-yet-another-pi-dma-hack-yapidh/details

[4] Source code of "librpicnc" library, https://github.com/binp-automation/librpicnc

[5] Source code of "pigpio" library, https://github.com/joan2937/pigpio

[6] Rust Programming Language, https://www.rust-lang.org/

[7] EPICS stress test scripts and setup, https://github.com/binp-automation/epics-stress-test

[8] EPICS device support for testing Cortex-A7 performance, https://github.com/binp-automation/devsup-template/tree/arm

[9] Rust for embedded devices, https://www.rust-lang.org/what/embedded

[10] EPICS bindings for Rust, https://github.com/binp-automation/epics-rs

[11] EPICS device support and driver for Keysight 53220A frequency counter written in Rust, https://github.com/binp-automation/ksfc-devsup

[12] EPICS device support template for Rust, https://github.com/binp-automation/devsup-template