ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPM080

SOFTWARE ARCHITECTURE OF FGC4, CERN'S NEXT-GENERATION POWER CONVERTER CONTROL PLATFORM

M. Cejp, D. Zielinski, R. Murillo-Garcia, CERN, Geneva, Switzerland

Abstract

The Function Generator/Controller is CERN's flagship controls platform for electrical power converters. Despite a proven track record, the current generation (FGC3) begins to show its age through performance limitations and component obsolescence. The requirements for its successor are ambitious: 100 kHz regulation rate (a 10-fold increase); reuse of a CERN-developed hardware platform (Distributed I/O Tier), improving synergy between CERN departments; and a software stack based on Linux with modern programming environments. The solution must fit CERN's accelerator control system, but also be fully usable at other institutions that use EPICS or TANGO through the Knowledge Transfer programme.

This paper discusses the software architecture, shaped by the need to separate real-time control processes from the Linux OS, which is achieved by dedicating separate CPU cores to each. Integration of CERN converter control libraries (CCLIBS) allows profiting from years of accumulated experience in the power converter domain. Results of performance characterization under different control scenarios are also presented, as well as lessons learned during integration in a test-bench environment.

BACKGROUND AND HISTORY

Electromagnets, sources and certain RF equipment in the particle accelerator complex at CERN are driven by electrical power converters. These converters are predominantly built in a modular way, with different teams designing the power electronics and the control system. Historically, a variety of controls solutions have been used, but in the early 2000s, a new universal controls platform was conceived, called the Function Generator/Controller (FGC). Although originally intended for the LHC, later generations are now increasingly being deployed in other accelerators at CERN.

The current generation, FGC3 [1], has been successfully used internally for a multitude of upgrade and consolidation projects (totalling over 2,100 devices deployed at CERN), as well as at external laboratories. However, the platform is starting to show its age through component obsolescence and performance limitations.

CERN Converter Control Libraries (CCLIBS)

The Converter Control Libraries [2] are a set of C libraries that aid in controlling the current, field or voltage in a circuit driven by a power converter. By creating libraries of the core control activities, the same code can be used in most FGC converter control platforms.

Besides their essential features, these libraries accumulate years of experience with power converter operation including subtle details in handling of various edge cases. It is therefore important to carry this previous work over to FGC4.

REQUIREMENTS AND CONSTRAINTS

The set of functional requirements is given implicitly by the capabilities of the current-generation system. However, certain demanding applications call for an increased iteration rate, up to 100 kHz.

The software architecture is shaped by two main constraints: the hardware platform chosen by the project, and the needs of CCLIBS (detailed below).

To reduce the cost of development and maintenance, the aim is to re-use existing software components as much as possible, and to minimize the amount of code in environments that are difficult to develop for, such as inside the Linux kernel or on "bare metal".

Hardware Platform

The hardware platform of FGC4 is the DI/OT System Board (Fig. 1), developed by the BE-CEM group at CERN. The central computing element is a Zynq UltraScale+ System-on-Chip (SoC). This SoC provides 4 CPU cores (ARM Cortex-A53) and programmable logic (FPGA) in one package.

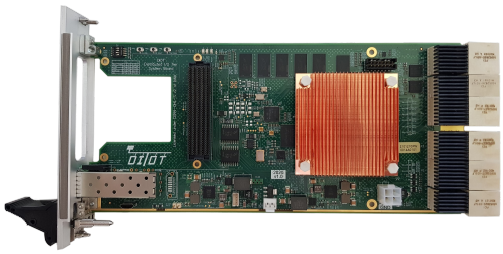

Figure 1: DI/OT System Board.

An important aspect of the platform is the amount of flexibility provided by the programmable logic. Historically, dozens of distinct FPGA designs have been developed to satisfy the needs of different types of power converters – often branching out from a main codebase with only minor changes.

The FGC4 departs from this model by providing a unified gateware design with an unprecedented degree of configurability exposed to the software layer. This design includes peripherals such as multi-channel ADC and DAC drivers, general-purpose I/O, timers, pulse generators and PWM generators. A typical power converter will consist of several electronic cards sharing a backplane, interconnected by multi-gigabit serial links.

ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPM080

CCLIBS Architecture

CCLIBS requires 2 independent threads of execution: a *real-time task* (RTP) and a *background processing task* (BGP). The real-time task is triggered at a fixed frequency in order to read up-to-date measurements, execute one iteration of a regulation algorithm, output a new actuation value, and carry out critical functions such as interlocks, limits and signal logging. The background processing task carries out activities which do not have a strict real-time constraint; one example is the arming of reference functions.

These two processes can share the same CPU, in which case the real-time task would be woken up by a periodic timer interrupt to take priority over the background processing. This approach, illustrated in Fig. 2, has been used in the current-generation FGC3. However, a split architecture is also conceivable, where the processes run on different CPUs (or different cores of a single physical CPU) and communicate via shared memory.

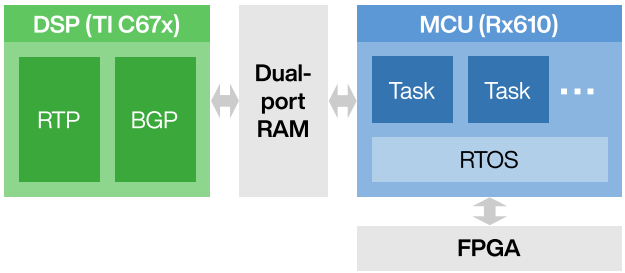

Figure 2: Multi-processing architecture of FGC3 (current generation). Performance and memory constraints dictate that real-time processing take place in a Digital Signal Processor (DSP) and non-real-time processing is split across the DSP and a general-purpose microcontroller.

DESIGN OF THE SOFTWARE ARCHITECTURE

Operating System and Environment

Linux would be a natural choice of operating system, not least because it is widely used across the Accelerator Control System. An important concern, however, is whether the system can provide the necessary hard-real-time performance: a 100 kHz regulation rate corresponds to 10 µs per iteration. If we assume that the processing takes, for example, 8 µs, this leaves 2 µs of margin for any scheduling jitter without overrunning the iteration period.

In a series of tests performed internally, with a heavily tuned kernel (PREEMPT_RT, CPU isolation, IRQ affinity, memory page lock), the achieved jitter figure was approximately 3.84 µs. While this is impressive for a Linux system, it was perceived as cutting too much into the iteration time budget.

As one possible alternative, the vendor provides support for FreeRTOS [3]. This real-time operating system is explicitly designed for environments where low and predictable interrupt latency is required.

Unlike Linux, FreeRTOS is not a standalone operating system with an interactive shell. Instead, it is always linked as an integral part of the application being deployed. This greatly reduces its footprint, but makes it more difficult to update the application after boot time. It also does not provide any facilities for recovery from program errors and crashes, or for debugging in general.

Finally, the main feature of FreeRTOS is providing multiple threads of execution with real-time guarantees. In our case, however, only one hard real-time thread is required. All other tasks would better be moved to Linux, which is an easier environment to develop for and to debug.

In light of these considerations, it was decided to dedicate one CPU core to execute the real-time tasks on bare metal (plus optionally a second core for a voltage control loop, when required for the given converter type), and moving all other responsibilities to Linux executing on the remaining cores, as illustrated in Fig. 3. This setup is also known as *asymmetric multiprocessing* (AMP). We further refer to a *bare-metal realm* and a *Linux realm*.

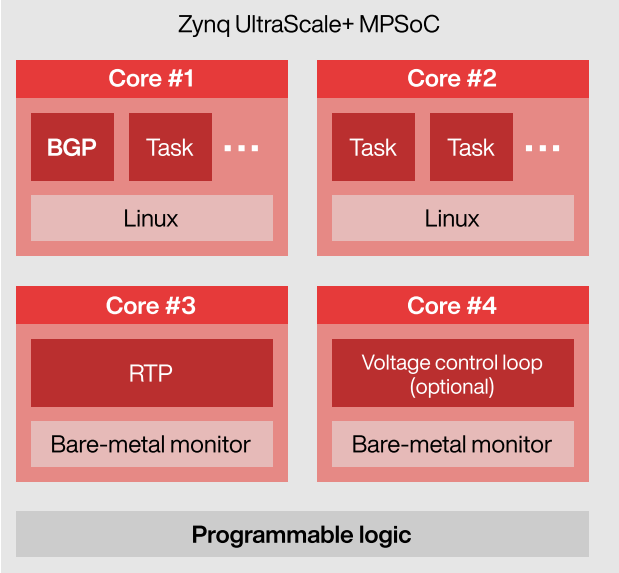

Figure 3: Multi-processing architecture of FGC4 (next generation). Real-time code runs on "bare metal" (with no underlying OS), while all other tasks execute in Linux user-space processes.

It is responsibility of the Linux realm to bring up the processor, and to load, start and monitor the real-time program. Afterwards, the program has full control over its CPU core. Occasionally it is necessary to "reclaim" control of the core, typically when a software update is requested, or if the program has crashed. For this purpose, a small *monitor program* was developed. This monitor program executes at a high privilege level (EL3), making it immune to corruption caused by errors in the main real-time program. It does not normally interfere with execution, but intervenes if triggered by an inter-processor interrupt (IPI).

ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPM080

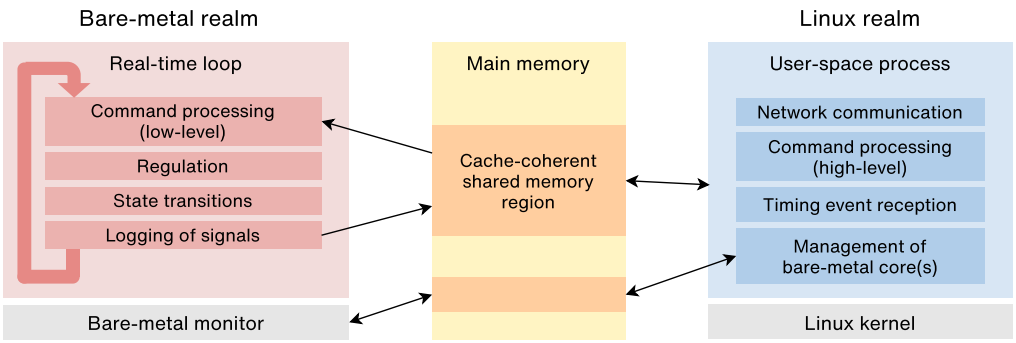

Figure 4: Task distribution and communication between the two realms. Only a subset of tasks in each realm has been chosen for illustration; the design of the user-space process (FGCDv2) is described in further detail in [4].

Cross-realm Communication

Figure 4 illustrates how the bare-metal and Linux realms are divided. The communication between the two relies on shared memory; a region of the memory space is dedicated for this purpose. Bare-metal programs access this region via its physical address while on the Linux side, the `/dev/mem` device is used to expose the memory to the userspace process via an `mmap` system call. This setup requires building the kernel with customized options, but it avoids the necessity to write a custom kernel module.

Cache coherency is an important concern in this type of architecture, since the individual cores each have their private L1 cache. The Cortex-A53 CPU includes a Snoop Control Unit (SCU), which maintains coherency of L1 caches across the cores. This eliminates the need to flush cache lines when exchanging data. For the SCU to work correctly, it is important that both cores configure their memory access attributes in a compatible way. Specifically, any shared pages must be mapped in a way that does *not* bypass the L1 cache.

Even with cache coherency, code must be written carefully to avoid creating race conditions among the different cores. In a conventional program, this could be solved by synchronization mechanisms such as mutexes. However, since the regulation code has a hard real-time constraint, it must never be required to wait for the Linux process which has much looser latency guarantees. Therefore, any data is always passed through lock-free data structures, such as double buffers or circular FIFOs.

As a concrete example, one essential feature of the FGC is to make the circuit current follow a waveform specified by the user (to *play a function* in FGC terminology). The generation of this waveform happens in the real-time process, but it is governed by a number of parameters that are remotely set by the user, which means that they are first received by the Linux realm. They must then be copied into a double buffer in the shared memory space. Next, the waveform is validated and "armed" for playback. Finally, it can be started by the user immediately, or synchronized to an external timing event.

PERFORMANCE

To ensure fitness for the application, the platform was evaluated in June 2022. Two realistic control scenarios were selected to represent different operating modes of CCLIBS. In one, a reference function is armed and played back; the other consists of smooth transitions between DC levels. The scenarios were executed on the ARM CPU. The achievable iteration frequency was then calculated by taking the reciprocal of the worst-case iteration time, as shown in Table 1.

Table 1: Benchmarking Results

	Scenario #1	**Scenario #2**
Mean iteration time	8.56 µs	8.46 µs
Maximum iteration time	10.52 µs	9.62 µs
Corresponding iter. rate	95.1 kHz	104.0 kHz

It can be seen that in the more demanding Scenario #1, the iteration rate drops below the 100 kHz target. However, the tests were carried out on a slower variant of the CPU: 1.2 GHz, as opposed to the 1.5 GHz final part. With this CPU upgrade, we expect to fit in the iteration time budget.

CONCLUSION

The FGC4 represents an important step forward in terms of converter control capabilities.

An important lesson was learned during the development of the bare-metal program. At the beginning, no hardware-assisted debugging was possible and the monitor program did not exist. This led to a slow development process, as it was often necessary to reboot the entire system due to programming errors crashing the CPU. The solution was to implement the privileged monitor, and to enable remote access to a JTAG debugging adapter connected to the development board.

The development of FGC4 is ongoing, with first prototypes to be delivered at the end of 2023. With the new capability of executing voltage control loops developed by other teams, FGC4 is not just a self-contained product, but an open platform that will continue evolving to meet our users' needs.

ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPM080

REFERENCES

[1] D. O. Calcoen, Q. King, and P. F. Semanaz, "Evolution of the CERN Power Converter Function Generator/Controller for Operation in Fast Cycling Accelerators", in *Proc. ICALEPCS'11*, Grenoble, France, Oct. 2011, paper WEPMN026, pp. 939–942.

[2] Q. King, K. T. Lebioda, M. Magrans de Abril, M. Martino, R. Murillo-Garcia, and A. Nicoletti, "CCLIBS: The CERN Power Converter Control Libraries", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 950–953. doi:10.18429/JACoW-ICALEPCS2015-WEPGF106

[3] FreeRTOS, https://freertos.org/.

[4] D. Zielinski, M. Cejp, and R. Murillo-Garcia, "Architecture Overview of the FGCDv2, CERN's Brand-new Power Converters Control Framework.", presented at the IPAC'23, Venice, Italy, May 2023, paper WEPM081, this conference.