FACILITY-WIDE SYNCHRONIZATION OF STANDARD FAIR EQUIPMENT CONTROLLERS

S. Rauch, W. Terpstra, W. Panschow, M. Thieme, C. Prados, M. Zweig, M. Kreider, D. Beck, R. Bär
GSI Helmholtzzentrum für Schwerionenforschung, Darmstadt, Germany

Abstract

The standard equipment controller under development for the new FAIR accelerator facility is the Scalable Control Unit (SCU). It is designed to synchronize and control the actions of up to 12 purpose-built slave cards, connected in a proprietary crate by a parallel backplane. Inter-crate coordination and facility-wide synchronization are a core FAIR requirement and thus precise timing of SCU slave actions is of vital importance.

The SCU consists primarily of two components, an x86 COM Express daughter board and a carrier board with an Altera Arria II GX FPGA, interconnected by PCI Express. The x86 receives configuration and set values with which it programs the real-time event-condition-action (ECA) unit in the FPGA. The ECA unit receives event messages via the timing network, which also synchronizes the clocks of all SCUs in the facility using White Rabbit. Matching events trigger actions on the SCU slave cards such as: ramping magnets, triggering kickers, etc.

Timing requirements differ depending on the action taken. For softer real-time actions, an interrupt can be generated for complex processing on the x86. Alternatively, the FPGA can directly fire a pulse out a LEMO output or an immediate SCU bus operation. The delay and synchronization achievable in each case differs and this paper examines the timing performance of each to determine which approach is appropriate for the required actions.

INTRODUCTION

In the FAIR control system, a data master issues high-level commands to control accelerator devices. The front-end controllers in the system react to relevant commands, issuing appropriate actions to their hardware components. Depending on the action to be taken, there are different timing requirements to be met.

Unlike the control system currently deployed at GSI, commands issued by the data master carry an absolute execution timestamp. The front-end controllers must receive commands early enough that they can schedule their actions to achieve the desired result at the correct time. Unfortunately, executing actions takes a variable amount of time. If the action takes 90-110 µs to execute, then this places two constraints on the system. Firstly, the data master must issue commands at least 110 µs ahead of time. Secondly, the system must be able to tolerate that the action could be as much as 10 µs too early or too late.

Issuing commands too far in advance reduces the responsiveness of the system. Once the data master has issued a command, it cannot be aborted. If the situation changes, perhaps due to interlock or contention from another beam user, the system cannot react faster than the slowest action already executing. This neglects, of course, other sources of latency in the system, such as network propagation delay, which only exacerbate the problem. It is thus generally desirable to have fast action execution.

Non-deterministic execution time is a potentially much more serious problem. For example, if a kicker executes an action a few nanoseconds too late, the beam might be lost. However, not all actions require the same precision, and it may make sense to trade accuracy for flexibility in some situations.

Fortunately, the most common equipment controller in FAIR, the Scalable Control Unit (SCU), has several possibilities for executing actions. This paper outlines the timing requirements of various accelerator components in FAIR and explorers the alternatives which could meet them.

USE CASES

The SCU will be the main frontend controller for the FAIR project. It provides a uniform platform connected both to the timing- and the data network of the facility. In turn, the SCU controls Adaptive Control Units (ACU) [1] slaves of various form factors which provide additional features and the necessary hard- and software interfaces to control the actual accelerator components. This means a wide range of magnet power supplies, Radio Frequency (RF) components and beam diagnosis equipment. The set of executable actions of course varies depending on the connected equipment. A magnet power supply for example will be provided with parameters and timed instructions to source a current ramp to its magnet, an RF generator gets different sets of frequency and phase parameters and the time when it needs to switch between them. The basic concept of the SCU envisions a complete separation of data supply and timing/commands. This way it can make use of higher abstraction levels, i.e. complex software, which brings flexibility, is more comfortable and maintainable, and also use low level hardware implementations to provide fast, deterministic behavior for the precisely scheduled execution of commands. The device controlled in the RF use case is called FPGA Interface Board (FIB) [2]. The kicker modules will be controlled by interface devices called IFK via a MIL-STD-1553 based field bus system.

SCALABLE CONTROL UNIT (SCU)

The SCU is mechanically a stack of up to three separated boards. There is the FPGA base board with an Arria II FPGA, two Small Form-factor Pluggable (SFP) slots, DDR3 RAM, parallel flash and a parallel bus (SCU bus)

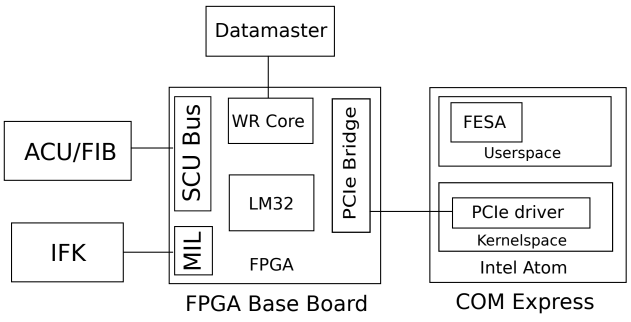

Figure 1: Block diagram of SCU.

for controlling up to 12 slave devices. In addition the base boards is equipped with White Rabbit [3] circuitry. A Com-Express module with an Intel Atom CPU is mounted to the base board. It has Ethernet, USB and PCIe interfaces. An optional extension board can be connected to the base board for backwards compatibility, that runs a MIL-STD-1553 based field bus interface.

The SCU works as a front-end controller. On one side it is connected to the control system via Ethernet, on the other side it controls slave devices over the SCU bus. It receives 1ns accurate timing information over a White Rabbit link, connected to an SFP. The White Rabbit receiver in the FPGA runs Precision Time Protocol (PTP) in software on a LatticeMico32 (LM32) soft-core CPU. The control system speaks to the Front End Software Architecture (FESA [4]) class running on the Intel Atom which is connected to the FPGA via a PCIe bridge.

EXECUTION ALTERNATIVES

When the SCU has an action to perform at a particular time, it has many alternative execution paths. Each option carries a trade-off between timing fidelity and the expressiveness of the program which performs the action.

FPGA

The SCU's FPGA can be programmed to generate the required output on a phase-aligned 8ns clock edge. When augmented by a fine delay card [5], this can be further improved to a general 1ns accuracy. The only source of non-determinism is the jitter of the FPGA's PLL (ps) and the inherent inaccuracy of White Rabbit (ns). Thus, both the delay and variability are in the sub-nanosecond range for this approach. Unfortunately, this execution path requires custom gateware and/or a simple output action.

LM32

Alternatively, the FPGA can issue an interrupt to an embedded soft-CPU (LM32). This on-chip CPU (with no operating system), can then run software to generate the appropriate action. The delay stems from the time to switch to interrupt context, run the software routine, and output the action. While broadly deterministic, cache behaviour and on-chip bus accesses contribute to runtime variability.

Table 1: Execution Timing Performance

µs	min	mean	max	stddev
FPGA	0	0.001	0.001	0.001
LM32	2.863	2.924	3.217	0.058
Kernel	7.120	13.29	37.73	3.49
Userspace	49.36	62.49	93.33	5.62
FESA	138.9	170.1	246.1	10.8

Atom-Kernel

Venturing further afield, the FPGA can issue an interrupt over PCIe to the Atom processor. The interrupt handler in the kernel driver then takes immediate action. This delays are the same as for the LM32, except that the interrupt is delivered off-chip via the PCIe bus. Furthermore, the Linux kernel may have interrupts masked in some critical sections, increasing the runtime variability.

Atom-Userspace

Rather than executing the action's software in kernel-space, the SCU could also deliver the interrupt to user-space. This adds additional context switch overhead, but provides for a more comfortable programming environment.

FESA

Finally, the userspace program which executes the action could use the FESA architecture [4]. Under this more general framework, the interrupt is translated to an action using multiple threads. This again increases the number of context switches and adds inter-process synchronization delay. However, it arguably provides the most flexible action execution framework.

ANALYSIS

To measure the delay and variability of the alternative execution paths, we connected two outputs from the SCU to an oscilloscope. The first output is the action aligned to the FPGA's 8ns clock. The second output is toggled by the execution path being measured. This approach excludes the variability of direct FPGA execution. However, this sub-nanosecond delay is dwarfed by the alternatives measured.

To capture worst-case variability, all test systems were subjected to a background work-load and include at least 10000 samples. For the LM32, we ran the white rabbit PTP core in the background, and performed a save/restore of all 32 registers on interrupt context switch. The Atom had a constant background task streaming text over ssh.

We tested the Atom with a real-time patched 2.6.33.6 Linux kernel. The PCIe bridge interrupt handler and kernel tasklet processes were set to real-time priority 99. For the userspace test, the test program had real-time priority 98. FESA set its own real-time priority to 60.

We also measured the LM32 without an instruction cache. This reduced the variability slightly from 354ns to 272ns, but greatly increased the average delay from

ISBN 978-3-95450-124-3

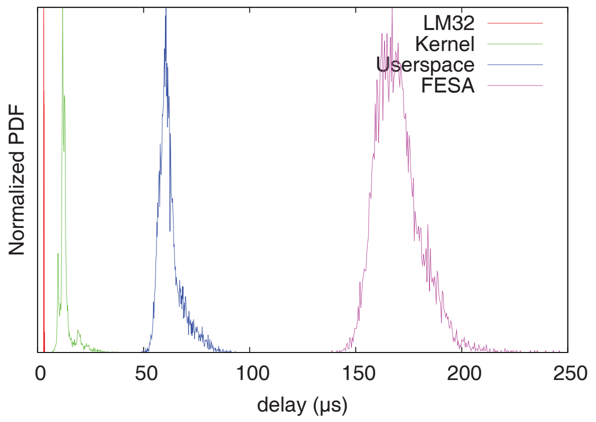

Figure 2: Comparison of delay distributions.

2.924 µs to 3.810 µs. Most of the variability appears to stem from pending Wishbone bus operations which can take multiple cycles to complete. Our LM32 was clocked at 62.5 MHz and when zoomed into the plot around 3 µs, it becomes obvious that the distribution has 22 spikes with 16ns intervals. When the background task was removed, this degenerates to 2 spikes, completely removing the variability. The 3 µs delay stems mostly from saving and restoring the register set. In principle, one could eliminate this cost using disjoint registers in interrupt and non-interrupt contexts.

From our measurements, both the time from interrupt to interrupt handler in Linux and the time from interrupt handler to userspace completion vary significantly. Unfortunately, with only 10000 samples, we could not trigger the worst case behaviour for both delays simultaneously. Adding the two worst-case delays measured separately, we predict that it should be possible for the Userspace delay to hit 120 µs and FESA 280 µs.

CONCLUSION

The measured times as presented in Table 1 must be reviewed in the context of different use-cases. As an example, ramping of magnets must be done synchronously. Here, a guaranteed synchronicity of 10-20 µs must be achieved for ring machines like the SIS18 and the SIS100. Another example is the control of kicker magnets, which requires at least 3ns precision and can only be done with FPGA Hardware Description Language (HDL). Software on the COM Express module may only be used for cases, where hard real-time is not required. As can be seen from the differences of minimum and maximum values, none of the solutions involving the CPU on the COM Express module fulfill those requirements, as long as the use of real-time Linux as operating systems is a stringent requirement for software tools like FESA.

For hard real-time the options are FPGA HDL or LM32 software. Here, FPGA HDL provides nanoseconds timing while LM32 software provides a better flexibility. To avoid stringent limitations for future developments of the FAIR accelerator complex, standard FAIR equipment controllers like the SCU should be designed supporting hard real-time on the nanoseconds scale. If flexibility during runtime is required, the ideal solution could be a combination of both options, where LM32 software creates the action patterns that are phase aligned with high precision by FPGA HDL.

REFERENCES

[1] H. Ramakers et al., “Adaptive Control Unit for Digital Control of Power Converters for Magnets in GSI and FAIR Accelerators”, GSI Scientific Report 2008, p. 117, `http://www-alt.gsi.de/informationen/wti/library/scientificreport2008/PAPERS/GSI-ACCELERATORS-14.pdf`

[2] M. Kumm et al., “Realtime Communication Based on Optical Fibers for the Control of Digital RF Components”, GSI Scientific Report 2007, p. 100, `http://www-alt.gsi.de/informationen/wti/library/scientificreport2007/PAPERS/GSI-ACCELERATORS-14.pdf`

[3] P. Moreira, J. Serrano, T. Wlostowski, P. Loschmidt, G, Gaderer, “White rabbit: Sub-nanosecond timing distribution over ethernet”, Precision Clock Synchronization for Measurement, Control and Communication, ISPCS 2009, DOI:10.1109/ISPCS.2009.5340196.

[4] M. Arruat et al., “Front-end Software Architecture”, Proceedings of ICALEPCS07, Knoxville, Tennessee, USA, p. 310

[5] `http://www.ohwr.org/projects/fmc-delay-1ns-8cha`