Facility-Wide Synchronization of Standard FAIR Equipment Controllers

Stefan Rauch, Wesley Terpstra, Wolfgang Panschow, Matthias Thieme,
Cesar Prados, Marcus Zweig, Mathias Kreider, Dietrich Beck, Ralph Bär

GSI, D-64291 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 reacts 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.

- commands carry absolute execution timestamp
- time limit for frond-end controllers for receiving commands
- depending on the time for processing an action
- tradeoff between responsivness and planning ahead

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

- main frontend controller for the FAIR project
- different slaves for different use cases
  - Adaptive Control Units (ACU) for power supplies
  - FPGA Interface Board (FIB) for Radio Frequency (RF) control
  - Kicker modules controlled by IFK via MIL-STD-1553 based field bus

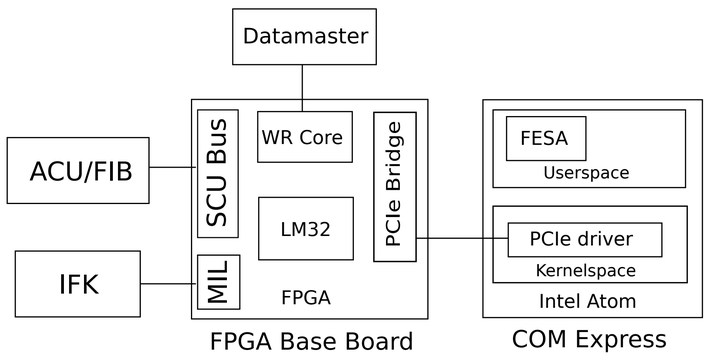

Figure 1: *Block diagram of SCU*

Scalable Control Unit (SCU)

- stack of up to three separated boards
- base board with Arria II FPGA, 2 SFP slots, DDR3 RAM, parallel flash, SCU bus
- White Rabbit Timing circuitry
- COMExpress module with Intel Atom CPU, connected via PCIe
- optional extension board for MIL-STD-1553 based field bus
- SCU receives 1ns accurate timing information via White Rabbit link

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

Figure 2: *Execution timing performance*

Execution Alternatives

- FPGA
  - can be programmed to generate output on 8ns phase aligned clock edge
  - with fine delay card down to 1ns
  - only source of jitter is PLL of the FPGA and inherent inaccuracy of White Rabbit
- LM32
  - FPGA triggers soft-CPU via interrupt
  - software generates appropriate action
  - delay from switch time to interrupt context and running the software
  - jitter from cache behaviour and on-chip bus access
- Atom-Kernel
  - FPGA interrupt directly handled in kernel
  - delays equal to LM32 + PCIe bridge delay
  - more jitter caused by Linux kernel
- Atom-Userspace
  - FPGA interrupt delivered to userspace
  - adds delay by context switch
- FESA
  - interrupt is translated to an action using threads
  - this increases number of context switches

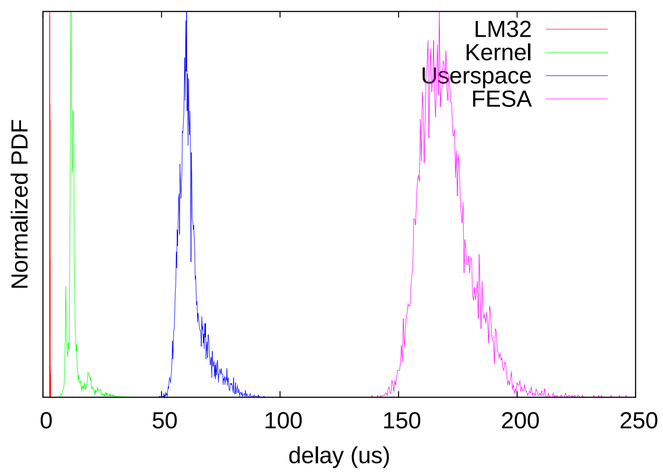

Figure 3: *Comparison of delay distributions*

Analysis

The outputs of an SCU were connected to an oscilloscope. First output is the action aligned to FPGA's 8ns clock. Second output is toggled by execution path. The approach ignores FPGA execution time. All test were done with background load and with at least 10000 samples. Load for LM32 was White Rabbit PTP core and save/restore of all 32 registers on interrupt context switch. The Atom was streaming text over ssh. Test were done with real-time patched 2.6.33.6 Linux kernel. The PCIe bridge interrupt handler and tasklet process were set to priority 99, the userspace test program to 98. FESA set its own priority to 60. We also measured the LM32 without instruction cache. This reduced the variability from 354ns to 272ns. Average delay was increased from 2.924μs to 3.810μs. Most of the variability seems to stem from Wishbone operations. The LM32 was clocked at 62.5Mhz and when zoomed into the plot around 3μs you can see a distribution of 22 spikes with 16ns intervals (Figure 4). The time from interrupt to the handler in Linux and the time from interrupt handler to userspace varies significantly. With 10000 samples the worst case delay was 240μs for FESA.

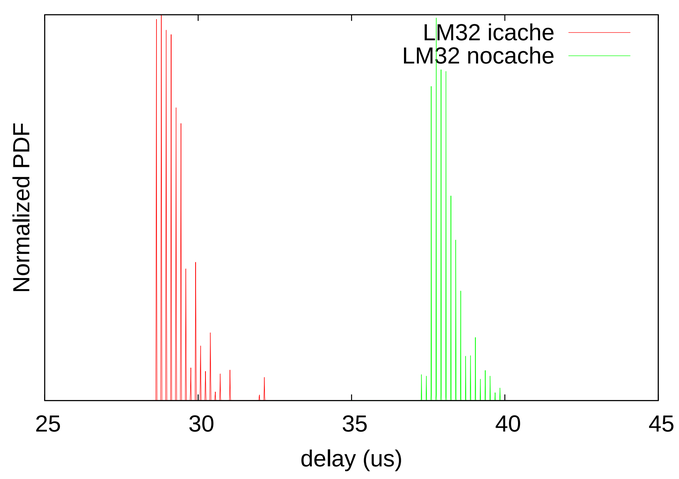

Figure 4: *LM32 delay distribution*

Conclusion

The measured times as presented in Figure 2 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 10-20μs 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. 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 to support 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.