THE FAIR TIMING MASTER: A DISCUSSION OF PERFORMANCE REQUIREMENTS AND ARCHITECTURES FOR A HIGH-PRECISION TIMING SYSTEM

M. Kreider, GSI Helmholtz Centre for Heavy Ion Research, Darmstadt, Germany
Hochschule Darmstadt, Darmstadt, Germany
Glyndŵr University, Wrexham, UK

Abstract

Production chains in a particle accelerator are complex structures with many interdependencies and multiple paths to consider. This ranges from system initialisation and synchronisation of numerous machines to interlock handling and appropriate contingency measures like beam dump scenarios. The FAIR facility will employ WhiteRabbit, a time based system which delivers an instruction and a corresponding execution time to a machine. In order to meet the deadlines in any given production chain, instructions need to be sent out ahead of time. For this purpose, code execution and message delivery times need to be known in advance. The FAIR Timing Master needs to be reliably capable of satisfying these timing requirements as well as being fault tolerant. Event sequences of recorded production chains indicate that low reaction times to internal and external events and fast, parallel execution are required. This suggests a slim architecture, especially devised for this purpose. Using the thread model of an OS or other high level programs on a generic CPU would be counterproductive when trying to achieve deterministic processing times. This paper deals with the analysis of said requirements as well as a comparison of known processor and virtual machine architectures and the possibilities of parallelisation in programmable hardware. In addition, existing proposals at GSI will be checked against these findings. The final goal will be to determine the best instruction set for modeling any given production chain and devising a suitable architecture to execute these models.

INTRODUCTION

In an accelerator, machines need to execute their actions at predefined moment, as part of an overall master plan. Magnets need to be run through current ramps, kicker assemblies need to be triggered at the right moment for beam transfer between accelerator rings, and similar matters. Most accelerators use some form of middle ware to convert physical scenarios into the necessary machine commands, like the production of a beam with a given energy and isotope. In theory, it would be possible to compute all necessary instructions and their times of execution in advance and then just go with this program. For such a scenario one would only need a very simple timing control, executing this precomputed value table.

However, more flexibility is needed in a real world scenario, since circumstances can change. From simply waiting for a machine to report ready status over changing demands from operators up to occurring interlocks and engaging machine protection circuits, all of these require changes to the planned scenario. Given a certain deadline, satisfying the timing requirements depends on several factors: The time to detect the condition and compose the message in the data master, transmission time through the distribution network, composed of switch traversal and time on the line, and decoding time in the endpoint.

TIMING CONSTRAINTS

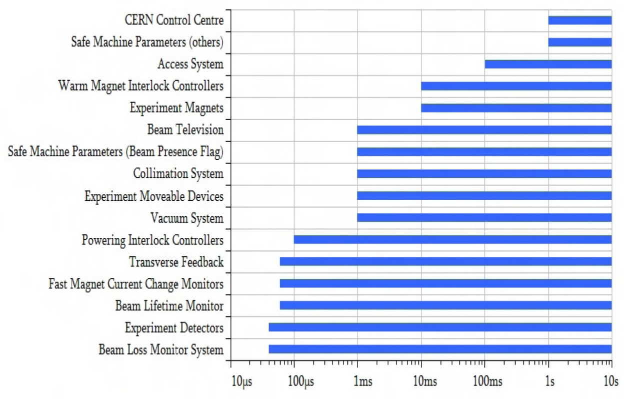

Figure 1: Reaction Times of CERN Interlock systems.[1]

The system timing constraints are to be deduced from hardware parameters like the minimum possible reaction time of a magnets power supply or the synchronisation requirements with RF systems and kicker assemblies as well as interlock systems, subdivided into beam protection, machine protection access control and operating. For all accelerators, interlock handling comes in several speed categories, depending on the kind of system to be monitored. from fastest to slowest these are: beam diagnostic equipment, magnet power supplies, vacuum system, temperature control, access control and finally operating.

Figure 1 shows a comparison between various systems employed at CERN's LHC. Interlock deadlines are spread here from 40μs over the low millisecond range up to several hundred milliseconds [2].

While many of these seem to require comparatively slow reaction achievable with a Real Time Operating System (RTOS), the fastest reaction times possible are always desired. The reason is, the quicker the system reaction time,

[1] B.Todd, A Beam Interlock System for CERN High Energy Accelerators

the less beam dump scenarios are to be expected. This means less ionisation of beam guides and Faraday cups, reducing radiation emmanating from the facility and therefore cooldown time in the beginning of maintenance shutdowns. It also causes less damage to beam diagnostic equipment. Availability for hands-on-maintenance goes with an overall decrease in maintenance time and cost. The most critical machine protection system shown here is the quench protection system for superconducting magnets. As to underline the importance of this system, the most prominent example is the 2008 incident at CERN. A quench led to the explosive discharge of about two tons of helium from a magnet housing into the LHC tunnel, damaging the equipment so severely, that the LHC was shut down for repairs until the end of 2009.

As an example from the planned GSI/FAIR facility, the shortest period to be served by the timing system is about 3ns for the kickers. This is obviously too fast for a reaction over the timing and controls network, but there is mixed approach to meet requirements and enhance system safety at the same time. The kicker timing depends on the RF frequency currently present on the accelerator ring, and this frequency is also sampled at the data master. Phase is therefore known to the whole system. After the kicker signals readiness, the event is scheduled to an absolute time 3ns after a future RF edge. The edge is chosen in a way to allow for transmission time of the control message trough the timing network plus a safety margin. Because transmission and processing time is somewhere around 100 μs, the kicker can safely be assumed to be unchanging in the interim. Until now, the kicker at GSI was triggered on a local basis without regard for other circumstances in the system, while said approach would shift back control to the timing master.

TRANSMISSION TIME

For an example of transmission and decoding times over a WR [1] network, we took a closer look at the future timing system of two accelerators, GSI/FAIR and CERN. The following estimates from the White Rabbit Robustness evaluation [3] show the time for transport of command messages on the network.

Table 1: Elements of Control Message Frame Delivery Delay Estimation [2]

Name	Value μs Min	Value μs Max
Eth Frame TX Delay	0	$(13 + B_{tx} t_{FECpck})$
Switch Routing Delay	0	13
Link Delay	$5[\frac{\mu s}{km}]$	$5\,[\frac{\mu s}{km}]$
Eth Frame RX delay	t_{FECpck}	t_{FECpck}
FEC Encoding	2	2
FEC Decoding	2	2

Both GSI/FAIR and CERN will employ a tree topology with three layers of WR switches between master and end-

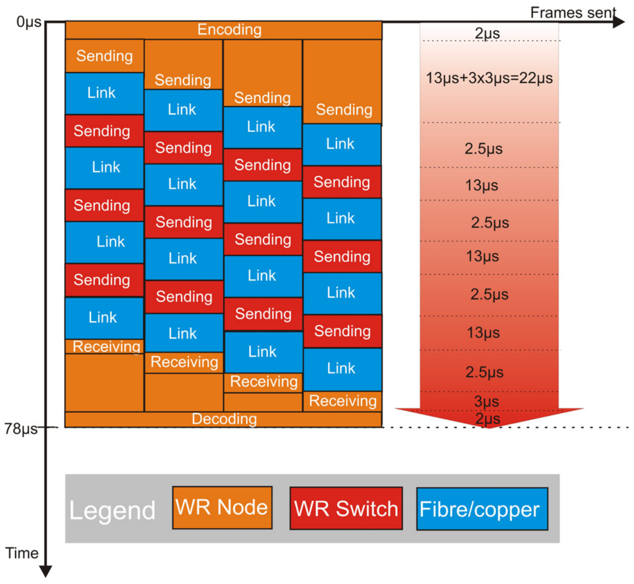

Figure 2: Delivery Delay of Control Message using cut-trough.[2]

points. The main difference between systems in the calculation is in the estimated length of fiber links used. The value is 2 km for GSI/FAIR and 10 km for CERN. The following values are calculated under the assumption that switches have a cut-through option to preempt command messages.

Table 2: Control Message Delivery Delay

Control Message size	Delay	
	GSI	CERN
500 bytes	78μs	118μs
1500 bytes	102μs	142μs
5000 bytes	162μs	202μs

The figures show that for interlock servicing, it would be advantageous to connect time critical devices to the upper switch layers in order to reduce transmission time. For fastest respone, connection to the topmost layer would reduce latency by about two thirds. From a practical point of view however, since the tree is fanning out, this will of course be limited by number of ports available on the first switch layer or even the timing data master.

MODELING THE ACCELERATOR

Command messages sent over the timing network supply event codes to the machines paired with execution times. The machines are preprogrammed with a certain reaction upon reception, which should also be done by the same means. The accelerator has sequential processes with synchronisation points. While the sequences are simple in themselves, their interdependencies can be quite complex,

[2] M.Lipinski, C. Prados, White Rabbit Robustness

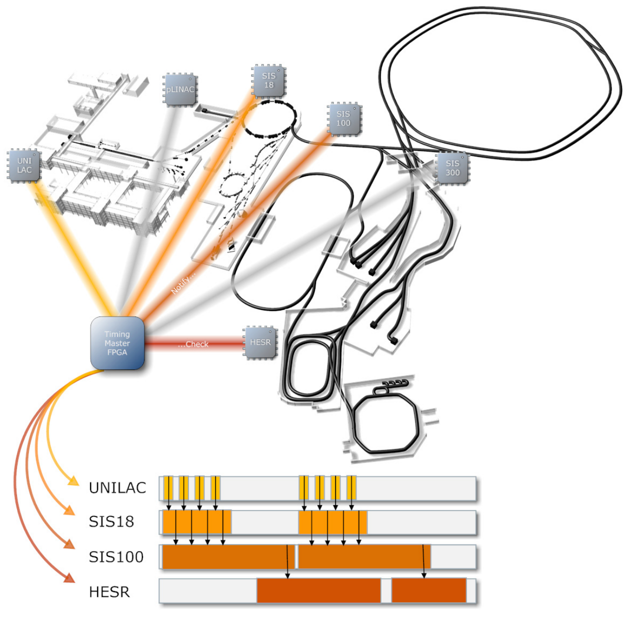

Figure 3: Machine sequences in a production chain.

depending both on external events like interlocks and internal synchronisation. Sequence architecture include jumps, conditional branching, IO handling, loops, real time clock waits and fast synchronisation between sequences. The sequences must be exchangable in parameters or completely during run time. For the GSI/FAIR accelerator, about 20 of these sequences are expected to run in parallel, 32 are aimed for to be future-proof.

PROCESSING TIME

CPU

With increasing complexity and number of processes, keeping code execution and synchronisation deterministic becomes ever more challenging. Fast modern CPUs, especially multicore architectures, offer vast processing power. Their drawback is that they require a managing OS to unlock most of their advanced features, which have worst case interrupt service latencies in the low millisecond range, making it an inapt choice for a Timing Master architecture. Real time Operating Systems are deterministic and can offer ISR latencies in the low microseconds range and most them are made for embedded CPUs (MCU). There are no RTOS systems targeting Hi-End-Multicore architectures available today.

Table 3 shows the latency measurements for a PowerPC 604 CPU (300MHz) on a MVME2306 board [4]. The discussed timing constraints could possibly be achieved with an MCU running an RTOS system, but as shown in Table 3, jitter is in the range of several microseconds. There are also strong peaks under load (which would be common, since the RTOS has to handle all machines and IO sources in the system) which go as high as 200μs. This disqualifies the solution for all fast interlock systems. We must also bear in mind that transmission time through the timing network ranging between 80-200μs must be added to the total reaction time, restricting the use even further. This leads to the assumption that an MCU with RTOS would probably be capable of running the required number of machine agents, but as figures show, could hardly cope with timing requirements for IO service requests.

Table 3: RTOS Latency Measurement Results[3] Times in μs

	Interrupt Latency		**Context Switching**	
	max	**avg ±**	**max**	**avg ±**
Idle System				
RTL	13.5	(1.7 ± 0.2)	33.1	(8.7 ± 0.5)
RTEMS1	14.9	(1.3 ± 0.1)	16.9	(2.3 ± 0.1)
RTEMS	15.1	(1.3 ± 0.1)	16.4	(2.2 ± 0.1)
vxWorks	13.1	(2.0 ± 0.2)	19.0	(3.1 ± 0.3)
Loaded System				
RTL	196.8	(2.1 ± 3.3)	193.9	(11.2 ± 4.5)
RTEMS1	19.2	(2.4 ± 1.7)	213.0	(10.4 ± 12.7)
RTEMS	20.5	(2.9 ± 1.8)	51.3	(3.7 ± 2.0)
vxWorks	25.2	(2.9 ± 1.5)	38.8	(9.5 ± 3.2)

FPGA Based Soft-CPU

This leaves programmable hardware for investigation. Field Programmable Gate Arrays (FPGA) can be configured by Hardware Description Languages (HDL) to function like every digital circuit from a simple FlipFlop over a counter to a full blown embedded CPU. The major advantage is in their huge degree of flexibility. There are several open source Soft-CPUs (Soft-CPU) as well as a couple commercial ones available today. Soft-CPUs in FPGAs are a fast, lightweight alternative to real hardware CPUs. Their most interesting aspect is that it is possible to have enough instances inside an FPGA to actually assign one Soft-CPU to each task. They also do not accumulate wait times for bus arbitration and memory access in operation, because they do not need to share their memory resources. A modern FPGA not only hosts logic units, but memory cells as well. Creating an instance of a memory controller template with matching RAM block for each Soft-CPU inside the FPGA makes them independent of their peers. In addition, this RAM block is natively capable of Dual Port access, providing the capability for on the fly change of sequence programs. Due to the fact of the one Soft-CPU-per-task policy, sequence programs can be blocking instead of using ISR, saving the time consuming context change. Assuming 125MHz system frequency and therefore 8 ns period per cycle, response times to external IO are well below 0.5μs, beating all competition. Due to the given flexibility, building a complete custom Soft-CPU tailormade for the given scenario is possible. The major drawback of a custom processing unit, wielding its own instruction set, is the total lack of toolchains. Virtually everything in a toolchain

[3] T. Straumann, Open Source Real Time Operating Systems Overview

that makes a productive workflow possible, that is, assembler, linker, compiler, debugger etc must be created from scratch. There will most likely be no community support, since the application is very narrow. Since the scenario requires standard functionality like basic math, counters, loops and RTC support, it will be a lot of redundant work.

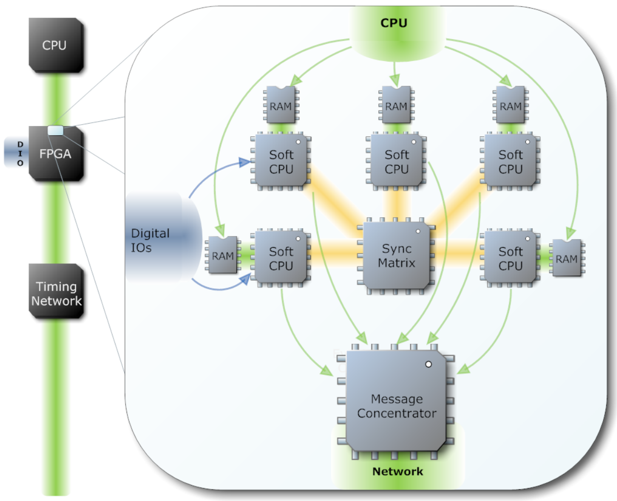

Figure 4: FPGA based Soft-CPU Cluster.

Soft-CPU Evaluation

A more promising approach is therefore to use a tried and tested open source soft CPU and enhance it with custom instructions. In our case, those are mainly ones used for quickly synchronising of soft CPUs.

Soft CPUs like the LM-32 are at around 2000 logic cells and therfore very lightweight [5]. Today's high end FPGAs range in the $>$ 250000 logic cells and can therefore easily instantiated several tens of those CPUs. According to early tests, the limiting factor is more likely to be the amount of RAM present in an FPGA than the number of logic units. Assuming around 2.5MB internal memory cells for a modern FPGA, this equals to about 80kB for each of the 32 softcores desired for the FAIR accelerator, more than enough to execute quite big programs. As an example, running a GNU debugger on system currently consumes around 1 kB memory.

The idea is then to create a Soft-CPU for each machine handler as well as one or more instances dedicated to IO signal and high level communication handling, as shown in figure 4. So instead of solving a complex scheduling problem in software to maintain deterministic system behavior, parallel tasks each get assigned their own Soft-CPU. Their synchronisation can be done by a custom HDL block, an $n * n$ matrix for synchronisation messages, making sync possible within less than five processor cycles ($<$ 40 ns) for checking of simple flag bits.

In the field of experimental electronics, a multi author situation should also be considered. It makes sense to let people working with the actual experiment write the software, which is another point in favor of existing toolchains.

CONCLUSION

In sight of time availabe to react to system interlocks and signals, it is evident that a pure CPU solution will not be able to satisfy the requirements. A pure custom hardware solution probably would satisfy the requirements for any given constraint, but prove difficult when it comes to follow changes at the operating level. Soft-CPUs are a good compromise between both worlds, with the added benefit of readily available development tools. To provide more IO speed and better scalability in face of system load, the choice falls to a multi core architecture from Soft-CPUs with dedicated internal memory. This setup shall be completed by a fast synchronisation mechanism and the possibility to dedicate Soft-CPU to specific sequence programs and IO handling.

OUTLOOK

The first steps will be the implementation of customized Soft-CPUs in a small scale test system in order to measure actual response times. An mechanism for safe on-the-fly change of the sequence programs in Soft-CPUs has to be devised, and several Soft-CPUs will be tested in a small cluster to demonstrate fast synchronisation between sequence programs. A prototype system is planned to be set up in 2012 to drive the first component to be comissioned to the FAIR extension of the GSI accelerator, a proton linear accelerator module.

REFERENCES

[1] P. Moreira, J. Serrano, T. Wlostowski, P. Loschmidt, G. Gaderer, "White Rabbit: Sub-Nanosecond Timing Distribution over Ethernet", IEEE Precision Clock Synchronization for Measurement, Control and Communication 2009, pp1-5, Brescia, October 2009.

[2] T. Straumann, "Open source real-time operating systems overview", Proceedings of the 8th International Conference on Accelerator and Large Experimental Physics Control Systems, San Jose, USA, 2001.

[3] M. Lipinski, C. Prados, "White Rabbit Robustness", 2011 `http://www.ohwr.org/projects/white-rabbit/repository/changes/trunk/documentation/specifications/robustness/robustness.pdf`, last visited 30.09.2011.

[4] B. Todd, "A beam interlock system for CERN high energy accelerators", Ph. D. thesis, Brunel University West London, 2006.

[5] W.W. Terpstra, "The Case for Soft-CPUs in Accelerator Control Systems", 2011, ICALEPS'11, THCHMUST05.