ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA092

TWO YEARS OF FAIR GENERAL MACHINE TIMING – EXPERIENCES AND IMPROVEMENTS

M. Kreider[1], A. Hahn, R. Bär, D. Beck, N. Kurz, C. Prados, S. Rauch, M. Reese, M. Zweig
GSI, Darmstadt, Germany
[1] also at Glyndŵr University, Wrexham, UK

Abstract

The FAIR General Machine Timing system has been in operation at GSI since 2015 and significant progress has been made in the last two years. The CRYRING accelerator was the first machine on campus operated with the new timing system and serves as a proving ground for new control system technology to this day. A White Rabbit (WR) network was set up, connecting parts of the existing facility. The Data Master was put under control of the LSA physics core. It was enhanced with a powerful schedule language and extensive research for delay bound analysis with network calculus was undertaken. Several form factors of Timing Receivers were improved, their hard and software now being in their second release and subject to a continuous series of automated long- and short-term tests in varying network scenarios. The final goal is time-synchronization of 2000-3000 nodes using the WR Precision-Time-Protocol distribution of TAI time stamps and synchronized command and control of FAIR equipment. Promising test results for scalability and accuracy were obtained when moving from temporary small lab setups to CRYRING's control system with more than 30 nodes connected over 3 layers of WR Switches.

INTRODUCTION

Motivation

The GSI-Helmholtz Center for Heavy Ion Research (GSI) in Darmstadt, Germany, is engaged in the ongoing development of a new type of control system (CS) for large physics experiments, which can utilize high accuracy timing. White Rabbit (WR), the underlying time synchronization technology, was an initiative started in 2008 by the European Center for Nuclear Research (CERN) and initially aimed at the modernization of the CS of the Large Hadron Collider (LHC) at CERN in Geneva, Switzerland. At the time, GSI took an interest in evaluating suitable technologies for a CS modernization and a new CS for the upcoming Facility for Antiproton and Ion Research (FAIR), a major extension to the GSI accelerator facilities. Research and development of WR successively became a close collaboration between CERN and GSI/FAIR.

System Layout

The FAIR CS approach aims for an alarm-based CS with clear separation of command dispatch and execution time. Furthermore, timed machine control and set-values supply use separate paths. This results in highly scalable system whose endpoints feature separate interfaces for configuration data, such as current ramps for magnets, and high accuracy timing and command, such as orchestrating the start times and successions of different current ramps. Figure 1 illustrates the concept: Settings Management derives both set-values and possible machine schedules from physical beam parameters. It provides specific set value data to individual endpoints (EP) to configure magnets, RF cavities, filters and other accelerator components (blue and red arrows). The hard real-time CS master, called Data Master (DM), is a deterministic scheduler employing high accuracy WR timing, sending commands to EPs in order to orchestrate the use of specific set-values at certain times (green arrows). The DM provides the flexibility to adapt the command stream to the facility's status, such as pending interlocks and beam requests, on the fly.

A hard-real time capable timing network deterministically distributes the DM's commands over a tree of fiber links and custom WR switches, while set-values are carried over standard gigabit copper network infrastructure. The timing network employs the deterministic Etherbone (EB) network protocol [1, p. 105-117] [2] to communicate with the EPs.

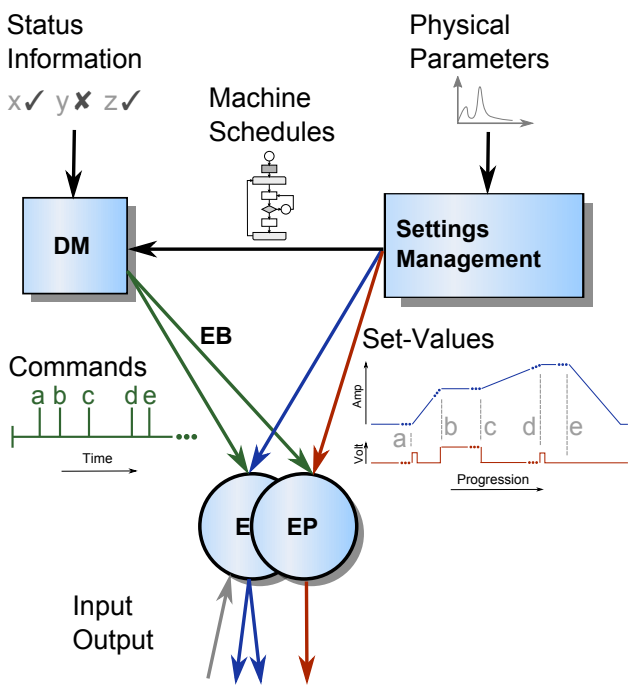

Figure 1: Schematic of FAIR CS. Left: Real-time Data Master (DM), Right: Settings Management Server. Endpoints (EP) receive both commands and set-values.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA092

Most EPs are, like the DM, hybrids of a programmable timing receiver (TR) hardware and a host computer running a Linux based operating system. Finally, EPs can use WR timing to time-stamp pulses on their inputs or assign time-stamps to sets of analogue data sampled by add-on hardware.

TEST IT TO BEST IT

In Verificatio Veritas

During the last two years, the necessity for test coverage of the complete CS stack became ever more noticeable. While the active WR community is a boon to FAIR CS development and GSI is itself an active contributor, it is a major challenge to keep FAIR TR designs up to date with current WR development progress. However, apart from consequent versioning, industry best practice can be hard to adapt to the small teams in research institutions, often required to practice agile development on top of a large and ever changing community code base.

Automated frameworks of standardized tests are an option that is much less likely to suffer from a developers white-box preconceptions. However, being generic, they also cannot be as thorough as bespoke tests written by verification engineers. But if the test intervals are small enough, the system offers developers close loop feedback to, alerting them when encountering bugs and strongly narrowing down possible causes.

Little Helper

To address the problem, a concept for a software driven, fully automated assembly and testbed for the FAIR CS was developed. Figure 2 shows the employed setup: Images are automatically created and tested on a small scale CS of multi layer WR network and TRs, their actions controlled by a DM. The current testbeds are capable of verifying individual FAIR CS components as well as testing their interplay and is able to verify timing accuracy down to 1 ns. While all shown components are functional today, they are currently split into two systems: CI automatically creates and image distribution for individual TR testbeds, the Timing Test facility uses the full CS stack for manually designed tests. Full automatic use of the full test CS is planned for mid 2018.

PROVING GROUNDS

For the past two years, the resulting prototype CS has been used to control CRYRING, a real world accelerator, providing valuable experience for control of the future FAIR facility, which is planned to go into operation in 2018.

CRYRING Accelerator

The CRYRING accelerator is a small synchrotron machine with its own linear injector stage. The most noticable successful test was CRYRING's first turn, that is, the beam achieving more than one orbit in the synchrotron ring. This success is documented in a FAIR press release from August 2016, where it was noted: "*Successful beam transport from ESR to CRYRING ... Furthermore, new beam diagnostics and FAIR-like control hardware and software could be tested with real beam.*" [3]

Experiments and Detectors

Contrary to equipment controllers, experiment data acquisition at GSI and the future FAIR facility intend to use WR TRs for mainly two purposes: Primarily, their time latch units are used to synchronize independent data acquisition systems. Extended tests have shown the system's suitability to seamlessly integrate globally triggered and free running data acquisition systems. A sampling time accuracy within 3 ns to 6 ns RMS could be achieved, which is sufficient to

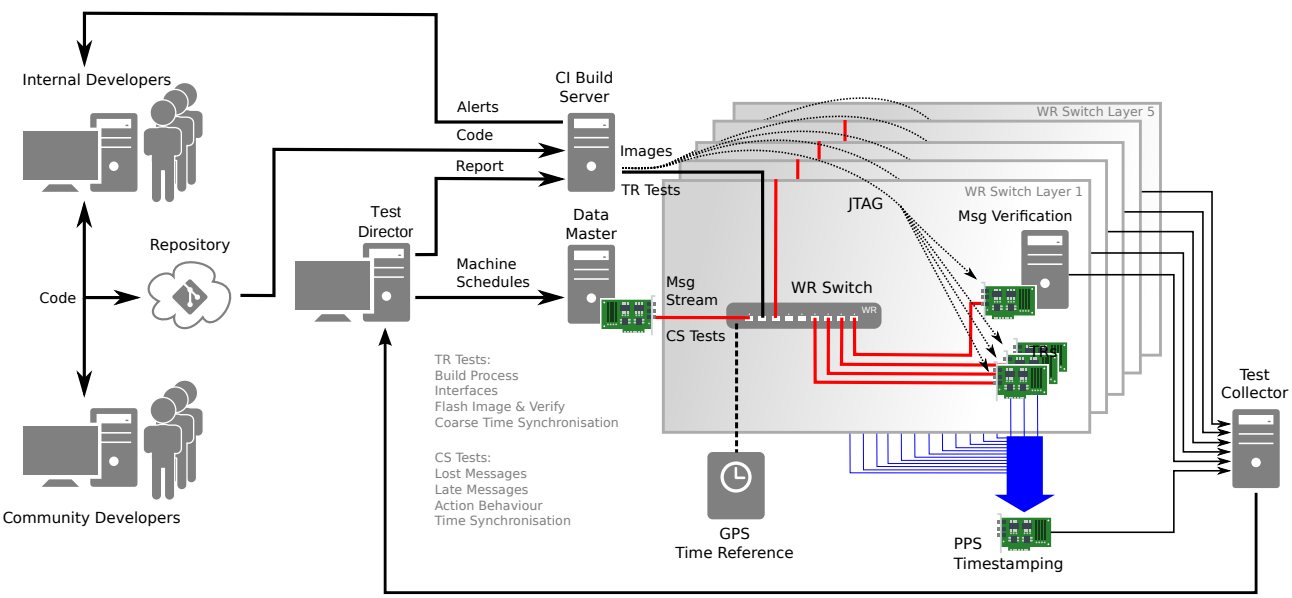

Figure 2: Schematic of the automated continuous integration system at GSI.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA092

correlate data from independent remote data acquisition systems at FAIR.

The second application is the use of synchronized clock outputs of WR TRs for distributed high precision time of flight measurements. Utilizing disciplined 200 MHz clocks from TRs, an overall precision of 11 ps RMS could be achieved as electronics resolution. This result indicates a the clock jitter for TRs of 5 ps to 7 ps when distributed campus wide in white rabbit timing networks. Both features were tested with the latest software release cherry and will be intensively used in the upcoming beam times.

Buffing the White Rabbit

GSI's timing team has also been working in close collaboration with Alessandro Rubini, creator of most of WR's firmware, to generally improve stability of the WR code and especially for Altera FPGA platforms. In addition, the team has developed and integrated sophisticated new WR logging and monitoring capabilities so as to increase the reliability of the synchronization.

A HEAD OF REAL-TIME

The DM version in use since 2015 acts as a message sequencer and offered a high level software interface to the settings management system, allowing it to be programmed during runtime. DM firmware was in turn designed to have deterministic behavior with as small a jitter as possible.

Meddling in Real-time: Command Queues

As it became apparent that the evolving FAIR use cases would require ever more real-time control over the DM, whole new set of problems had to be addressed.

New Rules on a Whim It was required that decisions at multiple points needed to be influenced from outside. This means specifying *who* should do *what* and *when* for each command, the *when* aspect being the most complex for software without the capability to share the DM's accurate notion of time.

Another problem was space. The DM's hardware memory inside the FPGA is very limited, so redundant schedules or commands would waste much needed space. But what is an appropriate size?

Further more, what in case of emergencies? On a beam dump alert, a fast reaction is required. No one would like the DM to execute queues of outdated commands, producing even more beam which needs to be dumped. So there also needed to be a way to preempt issued, but not yet executed, commands. The goals are therefore to allow multiple simultaneous points of decision, preserve command order, allow synchronization and preemption and, finally, have a space efficient repetition scheme. The conclusion was that each point of decision within a schedule must have its own command queue. These small software queues preserve command orders, but are designed as priority queues with three levels to allow preemption, i.e. if there is a command in high priority, it will be executed before any low priority command.

Generator Power To achieve the other goals, it could be shown that it is advantageous to send not commands, but *command generators*. In computer science, a generator is a function object which usually takes no further input parameters during execution. It uses an internal state to change its output behavior with each call. In this case, the inner state carries a time-stamp for synchronization, specifying when the generator is valid and should issue commands, and a counter, specifying how many outputs occur before the generator can be popped from the queue.

Intuitive Expression: Domain Specific Languages

Previous DM software versions used XML to describe machine schedules. The new demand for more flexibility brought this approach to its limits, as XML is not meant to model complex directed graphs. More research into the subject showed that graph based languages, such as GML or dot [4], are much more suitable to the task. In the end, dot was chosen for its wide acceptance and good integration with the Boost libraries [5]. Using custom node and edge properties, a domain specific language for machine control schedules was crafted, specifically for use with timing messages in the FAIR CS. One of the major advantages is that they do not need much textual overhead, and both their plain text and especially their visualization are easy to read for humans.

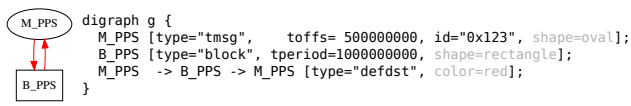

Figure 3: Minimal example for *dot* machine schedule.

DOT for FAIR CS Figure 3 shows the "hello world" PPS example, a basic machine schedule making TRs produce a pulse every second with a phase offset of 500 ms (cosmetic properties are grayed out). Nodes are connected by directed edges, there different types available for both. Time block nodes (rectangles) are time-spans, an auxiliary construct. Timing messages (drawn as ovals) are "real" content, they become messages on the timing network. A sequence of messages is always terminated by a time block, and each message has a time offset in nanoseconds relative to its corresponding time block. The DM also knows command nodes, which allow commands to be generated automatically from machine schedules. This enables automatic synchronization or loop behavior.

All traversed time blocks are added to a cumulative sum (one per DM thread), so a message's or command's deadline is simply calculated by adding its offset to the cumulative sum. Its dispatch time is in turn defined by deducting the planned time lead for transmission from its deadline. There are default paths through the graph which the DM will take

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA092

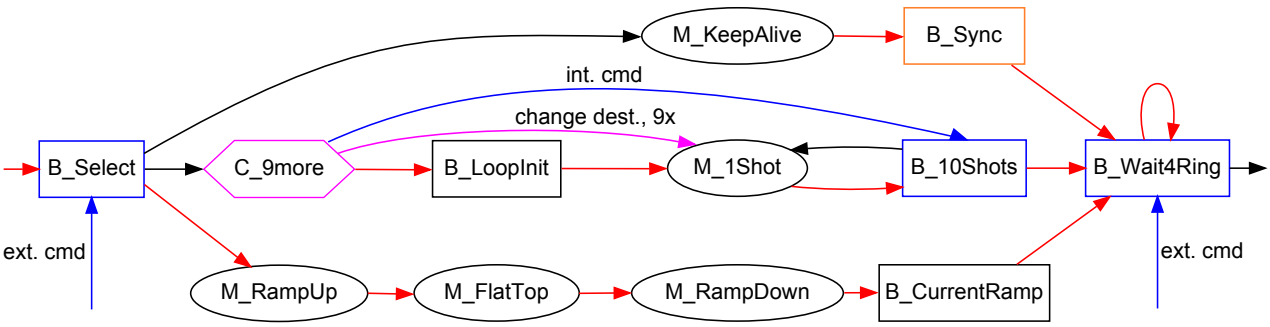

Figure 4: Visualization of *dot* demo schedule for DM functionality. Directed edges (default red, alternatives black) connect messages (ovals), commands (hexagon) to sequences terminated by time blocks (rectangles). Target (blue) and destination update edges (magenta) provide command context. Time blocks with command queues allow real-time changes (blue), sync time blocks auto-align to a given time grid (orange).

without outside interference (red edges) and alternative paths (black edges), which can be chosen by inserting commands into the corresponding queues. These queues are always attached to time blocks and add decision capability (blue rectangles).

CarpeDM To ensure a deterministic and fast real-time CS, the DM hard and firmware is completely unaware and unperturbed by memory management issues. Nodes and edges describe a graph that can be transformed to linked lists the DM can execute in real-time, each graph node is a fixed size (52 B) date node in the DM. The carpeDM software library on the host can produce the binary and also reconstruct the complete graph from it. This format is very space and time efficient on the DM hardware side, very few meta data nodes are needed to form command queues and list alternative paths. In addition, a bitmap is needed by the carpeDM, which is a small memory area with one bit set for each occupied data node to allow management. The native dual port RM allows parallel load without blocking CPUs. However, it is essential only sub graphs which are not currently in use are replaced. This is currently achieved by copying updated versions of graphs, but in future, path-finding algorithms like Dykstra's shall be used to check if sections allow hot in-place updates. Figure 4 shows a schedule with overly simplified context to show most of the available DM functionality: Multiple decisions, repetitions, wait loops and alignment can easily be programmed and visualized.

Safety First: Traffic Verification

The hardware design allowed the DM CPUs to be programmed without influencing the real-time behavior and processes outgoing messages with an earliest-deadline-first (EDF) scheduler. Together, these modules ensure the timely delivery of commands to timing receivers. But can they really, under all circumstances?

Knowing Your Limits Strictly speaking, the answer is "no". The outcome depends on the throughput and queuing behavior of the whole system, all components from the DM over the WR network down to the TRs. Once dispatched to the network, all messages are final. The time lead for message dispatch is therefore chosen as small as possible to ensure both timely delivery and provide maximum flexibility. Since the capacity and throughput of CS all components is limited, excess utilization will lead to late message arrivals. The goal must therefore be a system for calculating if the system will run safely with a given set of initial conditions. All valid machine schedule sets must therefore run within CPU utilization $\leq 100\%$, do not exceed the WR network bandwidth and maximum transmission delay does not exceed dispatch lead. The assessment is made even more complex by possible changes during runtime.

The Big Questions So is it possible to model the complex FAIR CS in a way that allows to obtain bounds for the maximum transmission delay for a given traffic flow? And if so, can a given *set of possible traffic flows* also be modeled, their combinatorial space expressed as a specification? Will that specification allow obtaining tight bounds for maximum transmission delay?

Calculating a Network

The last two years of research have shown that there are indeed ways to calculate feasibility in advance, even if real time control of the DM is allowed. Research in this field evaluating queuing theory and several other theories lead to the conclusion that there is a particularly suitable framework for providing guarantees for a communication system. Network calculus (NC) [6].

Introduction to Network Calculus NC is an approach that applies system theory to deterministic queuing systems found in communications, such as computer networks. Contrary to traditional system theory used for electronic circuits, NC employs a different set of algebra, the Min-Plus Dioid (addition becomes computation of the minimum, multiplication becomes addition). The approach is aimed at understanding and modeling fundamental properties of networks,

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA092

such as delay or buffer requirements, scheduling or window flow control, with a focus on worst case analysis.

NC represents network traffic as cumulative flows, i.e. the sum of data over time. Systems, such as network cards, switches, etc are usually defined by a maximum arrival curve and a minimum service curve. Arrival curves specify burst tolerance and maximum arrival bandwidth, service curves the system latency and minimum service bandwidth. Flows are shaped by arrival and service curves, and similar to filters in a signal path in system theory, the curves of simple NC systems can be combined to form more complex behavior.

Figure 5 shows the shaping effect of a system's arrival curve on a non-conformant traffic flow. The visualization also shows that at all points, the flow's latency and backlog can directly be deduced from the sytem's input and output curves.

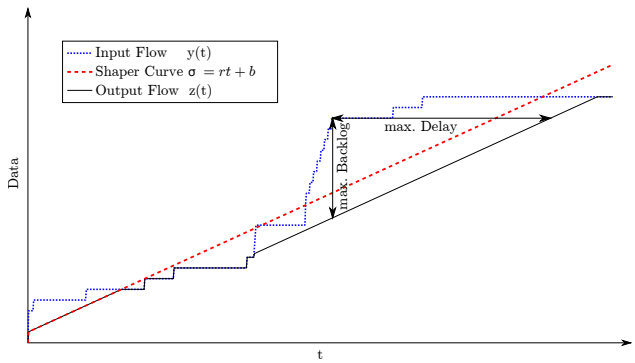

Figure 5: Network flow shaped by an arrival curve.

FAIR CS Model It could be shown that NC theory can provide a model for the FAIR CS network by modeling all of its subcomponents with arrival and service curves as well as scaling elements. A detailed explanation of the model and its background is out of the scope of this paper, but can be found in [1, p. 137-181]. The model and its curve parameters were then entered into the Disco DNC simulation tool [7], a numeric solver for deterministic network calculus, to calculate delay bounds for given a set of arrival curves.

Results showed that the FAIR CS can be guaranteed to stay within the target delay bound of 500 µs lead for the maximum net command bandwidth of 190 Mbit s^{-1}, but CPU load balancing and traffic burstiness must be individually analyzed during schedule preparation to prevent partial overload. The remainder of WR's 1 Gbit s^{-1} bandwidth is required for packet overhead, forward error correction (which is the biggest portion), and medium to low priority traffic such as time synchronization and management.

Curve Approximation Problem Transforming machine schedules to sets of possible input flows and further into tight arrival curves is itself no trivial matter and has only been lightly touched upon in the initial research as a proof of concept. This particular problem has recently been researched at the University of Kaiserslautern in collaboration with GSI, the result providing an algorithmic approach to obtaining tight arrival curves for FAIR machine schedules provided in the *dot* format. [8].

CONCLUSION AND OUTLOOK

In the last two years, we have gained valuable experience from productive use of the FAIR CS technology. This lead to various improvements in WR timing technology, TR SoC infrastructure and the CI verification scheme. It is planned to fully automatize the CI system in 2018, further improving our productivity and the quality of the CS. In addition, several new concepts allowing flexible real time control and traffic verification for the DM have been researched and implemented, going into test operation by the end of 2017.

Overall results were very positive, as the new FAIR CS equipment was able to cope with all use cases in the test phase on CRYRING. The next challenge will be testing the new improvements in the next release against FAIR use cases in 2018.

REFERENCES

[1] M. Kreider, "On Time, in Style: Nanosecond accuracy in network control systems," Doctoral Thesis, Glyndŵr University, Wrexham, UK, Aug. 2017. https://www-acc.gsi.de/wiki/pub/Timing/TimingSystemDocuments/kreiderPhdwiki.pdf

[2] M. Kreider, R. Bär, D. Beck, W. Terpstra, J. Davies, V. Grout, J. Lewis, J. Serrano, and T. Włostowski, "Open borders for system-on-a-chip buses: A wire format for connecting large physics controls," *Physical Review Special Topics - Accelerators and Beams*, vol. 15, no. 8, Aug. 2012. http://link.aps.org/doi/10.1103/PhysRevSTAB.15.082801

[3] FAIR, "First ring for FAIR," Aug. 2016. http://www.fair-center.eu/en/news-events/news-view/article/first-ring-for-fair.html

[4] E. R. Gansner, E. Koutsofios, and S. North, "Drawing graphs with dot," Tech. Rep., 2015. http://www.graphviz.org/pdf/dotguide.pdf

[5] J. Siek, L. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual, Portable Documents*. Pearson Education, 2001.

[6] P. Thiran and J. Y. Le Boudec, *Network Calculus*. Springer, 2001.

[7] S. Bondorf and J. Schmitt, "The DiscoDNC v2 – A Comprehensive Tool for Deterministic Network Calculus," in *Proc. of the 8th EAI International Conference on Performance Evaluation Methodologies and Tools (ValueTools)*, December 2014.

[8] M. Schütze, "Modelling and analysis of timing constraints of an industrial control system," Bachelor thesis, University of Kaiserslautern, Kaiserslautern, Germany, October 2017.