

FAIR TIMING MASTER

Mathias Kreider, Tibor Fleck, GSI Darmstadt, Germany

Abstract

In the scope of building the new FAIR facility, GSI will implement a new timing distribution system based on WhiteRabbit. The FAIR system will resemble a tree topology, with a single master unit on top, followed by several layers of WR switches, down to about two thousand timing receivers throughout the facility. The Timing Master will be a mixed FPGA/CPU solution, which translates physical requirements into timing events and feeds them into the WR network. Macros in the FPGA resemble a 32x multicore with a strongly reduced instruction-set, each event processor responsible for a specific part of the facility. These processors interact in real time, reacting to interlocks and conditions and ensuring determinism by parallel processing. A powerful CPU prepares the timing event sequences and provides an interface to the control system. These tables are loaded into the RAMs of each participating processor, controlling their behaviour and event output. GSI is currently working on the WR timing system in close collaboration with CERN, making this system the future of GSI/FAIR. This contribution covers technical details on the expected timing scenario, macro internals and discussion on possible future development.

INTRODUCTION

Purpose

Future GSI/FAIR facility will use timing events to control machine actions. The FAIR Timing Master will centrally generate all necessary events for the whole accelerator facility. These will be used to trigger all beam guiding components as well as all beam diagnostic measurement devices where individual event filters apply for each single front end controller. The timing receiver is integrated into the standard FAIR frontend controller used mainly for power supplies. For all other use cases, especially all beam diagnostic devices, special timing receiver interface cards will be supplied in different form factors. Typical event reaction will be direct trigger output or IRQ. Furthermore a separate high precision clock distribution system called BuTiS for RF components where highest requirements to accuracy and synchronization apply will be closely coupled to the FAIR timing system.

The WhiteRabbit Transport Layer

The future Timing System of GSI/FAIR and CERN will be based on the WhiteRabbit architecture. WR is a deterministic field bus [2], the physical system consists of a non-meshed GbE network topology, running timing services on Control hardware and low-level software

OSI layer II. Custom switches and endpoints are used for timing measurements and the WR protocol.

WR provides phase compensation and absolute time distribution with an accuracy down to a nanosecond. Forward error correction algorithms are employed to get highest system reliability. Deterministic lag times are made possible by using Quality of Service (QoS). This makes preferring marked high priority packets possible. Since the lag time to destination is reliably known in advance, this allows machine control packets to always arrive on time.

The FAIR Timing Master

To provide an interface to the general control system of the facility, a powerful CPU handles the abstract beam production down to the creation of sequence programs for control of Event Processing Units (EPU).

Every abstract physical part of the accelerator facility like the linear accelerators, synchrotron rings and storage rings, will be represented by a dedicated timing event generator unit.

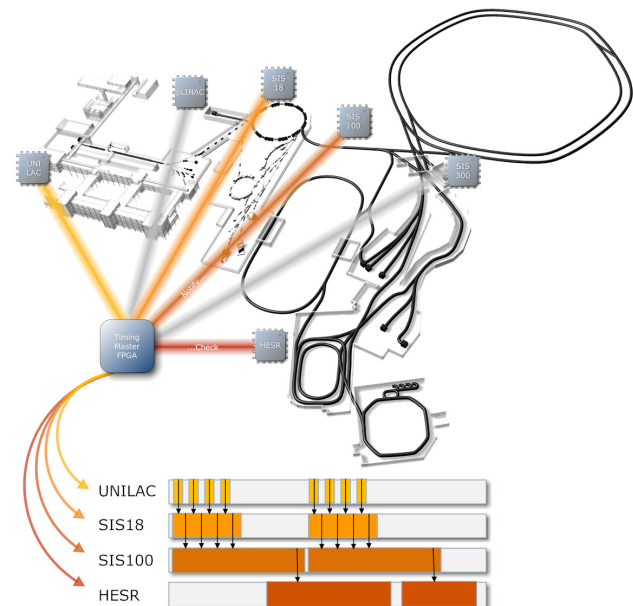


Figure 1: Mapping components to EPU programs

Interaction between these machine parts requires fast synchronisation between their timing schedules. For example, a synchrotron ring needs to time its ejections precisely with the receiving collector ring. The design therefore includes a fast mechanism for exchanges between generators.

Control solutions with FPGAs

PLANNING

The Timing Master (TM) is a mixed approach between a powerful CPU for easy integration into operating software as well as easy use of existing libraries and middleware. However, CPUs have underlying core and power management functions which make response times in the desired range unpredictable. An FPGA is used for event generation and time critical communication.

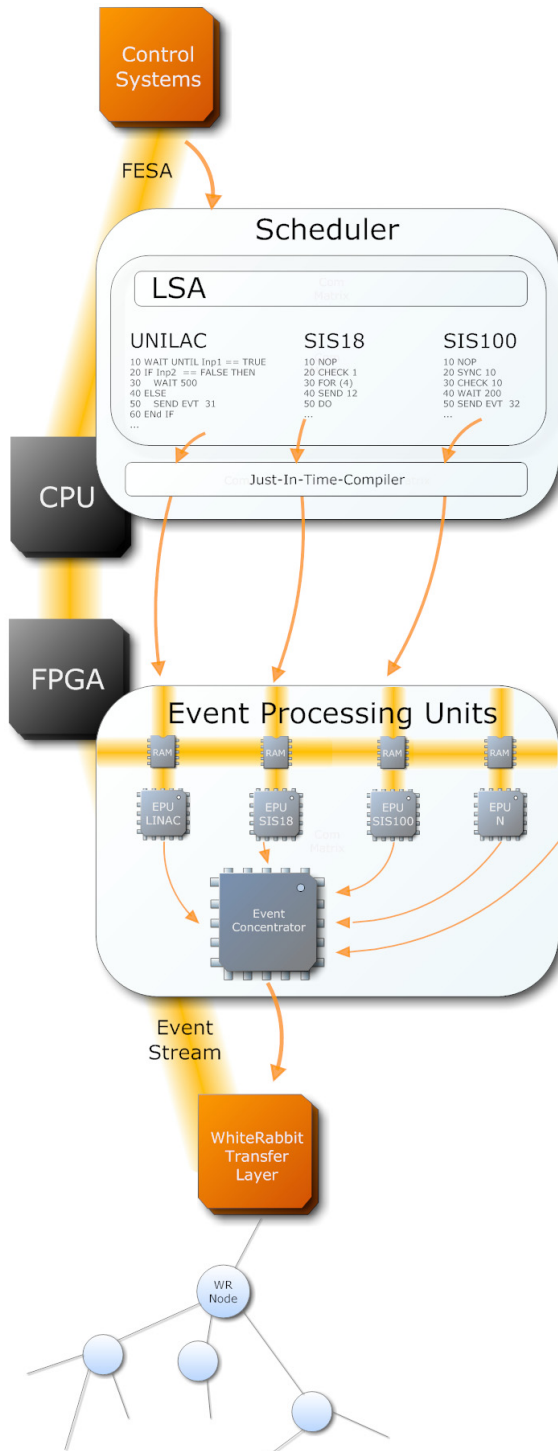


Figure 2: Timing Master Functional Blocks

Control hardware and low-level software

ARCHITECTURE

Figure 2 shows the details of the current implementation in testing. It shows the data flow from Operating through machine translation, conversion to programmatic format and real time execution inside the FPGA submitting data to the transfer layer.

DATA FLOW

The TM's CPU gets instructions for a production line (Isotope, Amount, Energy, Source, Path, Target) from Operating. Physical requirements are translated into machine requirements by the LSA middleware. The resulting event sequences with their dependencies are transformed into event generator programs, compiled and loaded into the FPGA's memory where they will be executed.

IMPLEMENTATION

CPU

The top interface to the control system is based on FESA, a device model framework and driver package. An interface to the LSA core provides machine behavior descriptions calculated from physical parameters. Below this is the event sequencer, compiler and FPGA communications module. As soon as a production line is fully defined, relative execution time of all necessary events and dependencies between the indiv To allow reload of new programs during runtime without interruption, memory write access is managed by CPU only to write in places not currently locked for execution. Preinserted conditions in the EPUs program allow branching off to new program code on demand. Sequences for all interlock, beam abort or beam request scenarios can be predefined for all EPUs. The timing masters real-time decision logic will then always switch to safe, consistent alternative event sequences.

FPGA

The FPGA houses multiple processor macros, each with its own memory and controller. These Event Processing Units (EPUs) are all equal in implementation, their behavior is fully determined by the programs loaded into their RAM.

EPUs are made dedicated to certain parts of the facility or serve public functions, like collecting and processing beam requests from experimental stations. This has the benefit of having a human readable program for each timing generator. and interaction between involved components is easily traceable because it follows the supposed beam path. This leads to well defined modules and interfaces, which can easily be tested standalone, therefore speeding up system development.

Their programs run in parallel, are able to listen to external signals and interlocks, generate timing events and synchronise themselves with other processors by an n by n

Control solutions with FPGAs

flag matrix in a single cycle. In order to achieve fast synchronisation, the flag matrix (each EPU can signal all other EPUs) is completely realised as FlipFlops for fastest access. An EPU can set its own flag vector and read the Bit concerning itself from all other flag vectors, clearing it in the process.

EPU and Instruction Set

The EPUs opcodes define mainly programmatic courses, like jumps, branches and nested loops. They are not general purpose processors but specialised sequencers. An EPU instruction contains an Opcode, IO select instructions and dedicated data fields for event codes, time values and constants. This is not an optimal use of the FPGAs memory, but certifies execution times for each opcode and completely circumvents memory fragmentation.

WR Interface

For issues of load balancing, the Timing Master will have a 100 μ s collection cycle or granularity window for outgoing events. The event concentrator macro then sends a compact stream of events to the WR module, where they are channel encoded and grouped into Ethernet packets. Packet size also has an impact on the effectiveness of the Forward Error Correction algorithm used in WR [4]. Current settings expect a packet length of at least 200 byte for the FEC to work efficiently, otherwise padding bytes must be added.

Since the Timing Master broadcasts all events facility-wide and only a few events are valid for an individual node, predefined event filters will run in each nodes FPGA. When an event is received, a node typically issues special trigger signals or interrupts.

CONCLUSION

The concept of dedicated EPUs representing accelerator components showed promise in early simulations.

A small number of EPUs were run with hand written test programs, covering scenarios with up to four cooperating EPUs. The task at hand is scaling these scenarios in simulation to copy real scenarios. As soon as the simulation is able to reproduce slowed down event sequences of the current controls system, modules will be prepared for synthesis.

OUTLOOK

A prototype system is planned to be set up in parallel to the current pulse centre in 2011. By comparing control sequences, a continuous test for aptability to the task of running the current facility can be done. First test is run with pre-written event programs, this allows testing in productive conditions without further concern about scheduling and machine calculations done above or transfer down below.

Control hardware and low-level software

After a first design stop of the EPU macros, the next goal is an early implementation of the TMs software modules most importantly the Event Sequencer and compiler. The sequencer will be a solver tool able to synthesise the LSA output sequence by reducing it to programmatic structures and event numbers. The current compiler for the EPUs language can be converted to a JIT-Compiler module for the master.

A productive system is planned to be put into service at GSI/FAIR in 2016.

REFERENCES

- [1] P. Moreira et al., "White Rabbit: Sub-Nanosecond Timing Distribution over Ethernet", ISPCS 2009, Brescia, Italy, Oct 2009
- [2] J. Serrano et al., P. "THE WHITE RABBIT PROJECT", TUC4 ICALEPS2009, Kobe, Japan, Oct 2009
- [3] WR Switch Specifications <http://www.ohwr.org/>
- [4] C. Prados Boda, T. Fleck, "FEC in Deterministic Control Systems over Gigabit Ethernet", THPL011 PCaPAC2010, Saskatoon, Canada, Oct 2010

Control solutions with FPGAs