# FAIR Timing Master

M. Kreider, T. Fleck

*GSI, Darmstadt, Germany*

FAIR

## Abstract

In the scope of building the new FAIR facility, GSI will change its timing system to 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 realtime, reacting to interlocks and conditions and ensuring determinism by parallel processing. A powerful CPU prepares the timing event tables and provides an interface to the controls 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. The poster will cover technical details on the expected timing scenario, macro internals and discussion on possible future development.

## Goals

- Provide subnanosecond Timing
- Control Machines by Events
- Manage 2000+ Timing Nodes

## Concept

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

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 mechanism for exchanges between generators.

## Planning

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

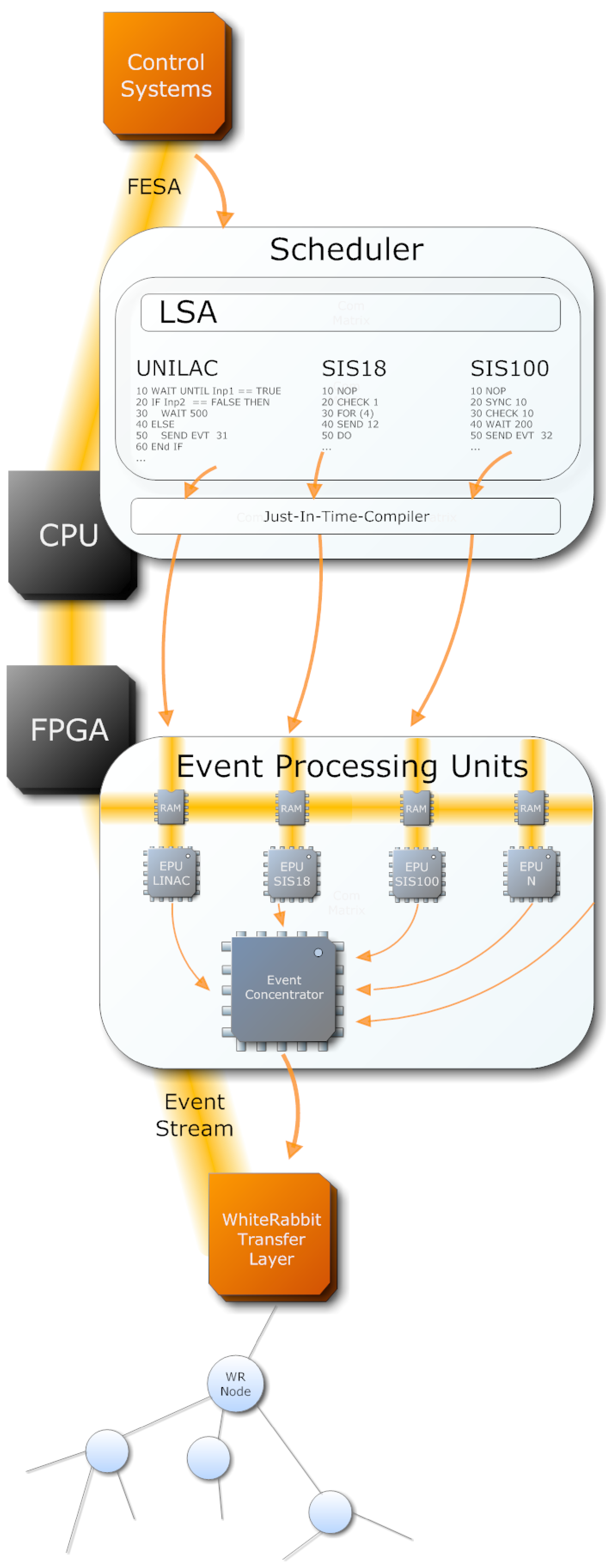
## Implementation

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

Inside the FPGA, the event processing units (EPUs) are dedicated to certain parts of the facility or serve public functions, like collecting and processing beam requests from experimental stations. 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 flag matrix.

## Modules and Interfaces

The TM CPU houses two important pre-existing software modules. The top interface to the control system is based on FESA device models. Beneath the driver layer is an interface to the LSA core, which computes machine parameters from beam production specifications. Below those are the event sequencer, compiler and FPGA communications module.



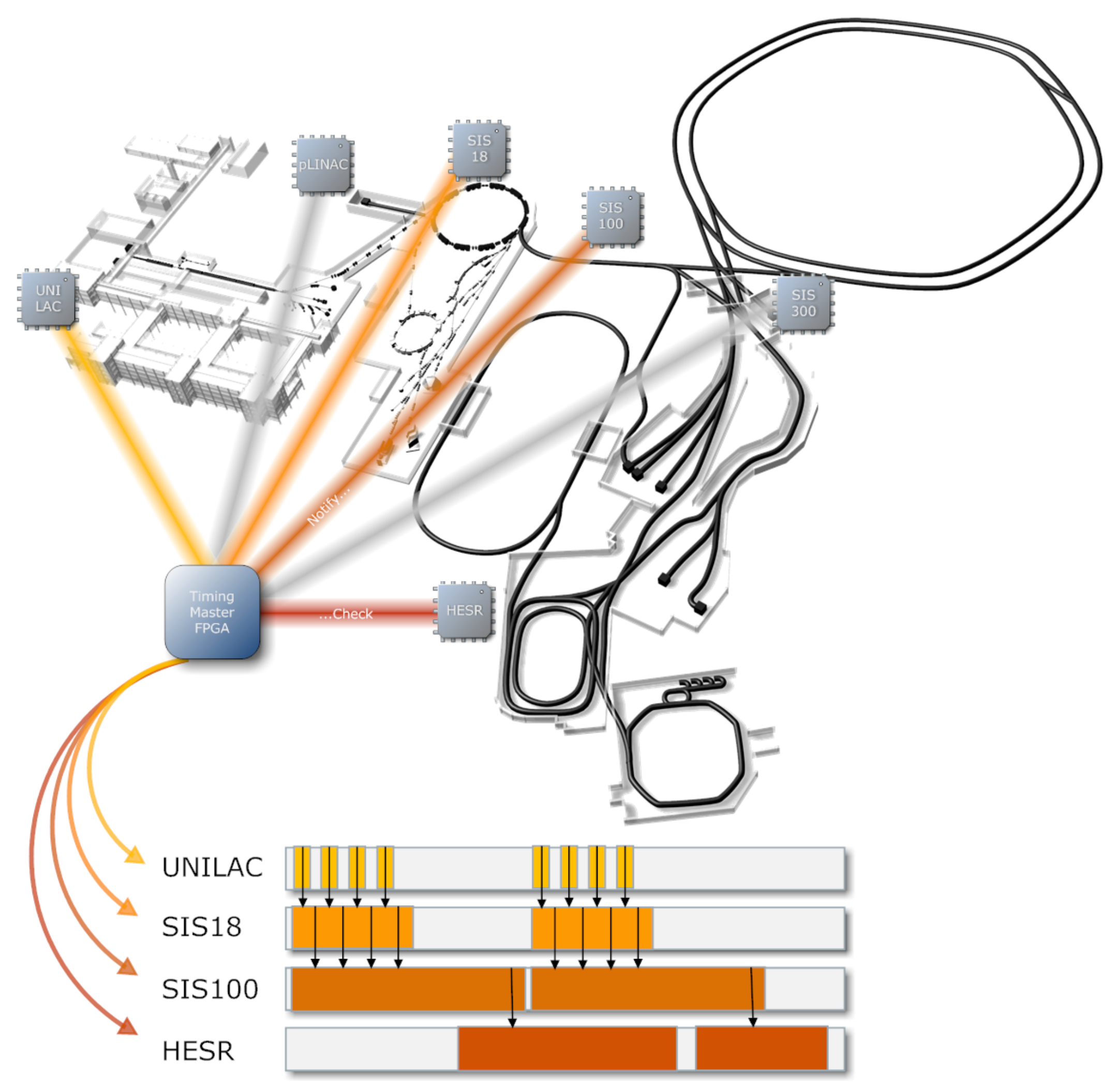
```
UNILAC
10 WAIT UNTIL Inp1 == TRUE
20 IF Inp2  == FALSE THEN
30    WAIT 500
40 ELSE
50    SEND EVT  31
60 ENd IF
...

SIS18
10 NOP
20 CHECK 1
30 FOR (4)
40 SEND 12
50 DO
...

SIS100
10 NOP
20 SYNC 10
30 CHECK 10
40 WAIT 200
50 SEND EVT  32
...
```

*Overview of Hard- and Software Modules*

The concentrator macro collects all given events in a certain window of time and passes the event stream to WR for transfer.

## Mapping the Accelerator

Timing nodes of the facility part are listening to a specific subset of eventcodes usable for their control sequence. On reception, the listening node sets up the time of execution and starts the corresponding process.



*Linked EPUs building an event sequence*

To allow reload of new programs during runtime without interruption, memory 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.

## Conclusion

The concept of dedicated EPUs representing physical accelerator parts 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 verify real scenarios. As soon as the simulation is able to reproduce slowed down event sequences of the current controls system, modules will be pepared for synthesis.

## Outlook

A prototype system is planned to be set up in parallel to the current GSI 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.
After a first design stop of the EPU macros, the next goal is an early implementation of the TM 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.
Productive systems are planned to be put into service at GSI/FAIR in 2016.