Timing Receiver: Event-Condition-Action (ECA) Unit Design[1]

(W. Terpstra, ver. 23-May-2013)

In the FAIR timing system, the data master (DM) prepares the concrete schedule of commands which will be executed by time-sensitive component devices. Each device is controlled by a front-end equipment controller (FEC). Such a FEC includes a Timing Receiver (TR), which receives and interprets the command schedule from the DM. In order to realize precise coordination between devices, the TRs embedded in the FECs include hardware to synchronize their clocks. The DM leverages these clocks by sending absolute execution timestamps with its commands.

Given the large number of devices in the FAIR project, the DM's commands do not directly describe the actions taken by a FEC's device(s). Instead, the DM operates on a logical/task level with commands such as ramping a particular accelerator ring. An accelerator ring is controlled by many component devices, each with their own FEC. Commands are broadcast by the DM to the FECs using *timing messages.* Timing messages may contain one or more *incoming events,* which are locally interpreted by the FEC's TR to produce *actions* at the designated time. As the behaviour of each TR differs depending on its role in performing the logical command, TRs are programmed with a *condition* table that describes how to process events. The TR component which translates these incoming events to actions is called the event-condition-action (ECA) unit.

The key advantages of the indirect ECA approach over direct control by the DM:

1. The DM need only concern itself with scheduling high-level accelerator commands.
2. High-level events provide a better overview when debugging the DM control system.
3. There are less commands for the DM to transmit, essentially compressing the stream.
4. Calibration and configuration of TR can be done locally, without involving the DM.
5. The system is more modular; FECs including their Trs can be added/removed without changing the DM.

The outputs of the ECA unit are the action *channels*, each attached to a controlled interface. Channels output actions at the absolute time appropriate to meet the DM's command schedule. A recipients of an action channel is a *receiving component* (RC) inside the TR's FPGA. The task of a RC is the generation of so called *timing events* at a FEC. Examples of timing events are host interrupts, digital pulses, or SCU bus requests. The particular behavior of the *receiving component* (RC) attached to an action channel lies outside the scope of the ECA unit. In FAIR, the RCs might be configured by FESA with set values prior to receiving the action/timing trigger from the ECA. However, this is strictly optional.

Please refer to the glossary at the end of this document!

Core requirements for the ECA unit:

ECA-1: *Local timing offsets*

The absolute timestamp provided by the DM in a timing message is modified by the ECA unit according to its condition table. For example, two FECs might need to ramp power supplies with different response times. During installation, a local offset is configured which the ECA will add to the absolute event timestamp to create the absolute action timestamp. Thus, when

1 Please note the difference between *incoming event* and *timing event*.

the DM issues a ramp command, it need not know the local calibration needed by each FEC. Remark: An offset might also be negative. The timely delivery of timing events is only guaranteed, if the offset is within an allowed range which is defined by the timing team.

ECA-1.1: *Past incoming event detection*

In the case that an incoming event is translated into an action which should be taken in the past, the ECA unit posts the action with an error indication, reporting the incoming event which caused the problem. Naturally, it is the responsibility of the DM and the timing network to ensure this never happens. When a past incoming event is detected, the status flag should be cleared and reported by the FEC's host, potentially for triggering post mortem.

ECA-2: *One incoming event, multiple channels*

A single incoming event from the DM may require actions at more than one interface. For example, a ramp event may require both controlling the power supply and signaling an interrupt for the host system.

ECA-3: *One incoming event, multiple actions (per channel)*

A single incoming event from the DM may require multiple actions on a single interface. For example, an incoming event which generates a digital pulse requires an action to turn the digital output both on and (later) off.

ECA-4: *Simultaneous actions out multiple channels*

The ECA may need to issue actions out several interfaces simultaneously. These actions may have been spawned from the same incoming event or two unrelated incoming events.

ECA-5: *Action collision detection*

The ECA can only deliver a single action per channel per 125MHz interval. Thus, the timing team must carefully configure the DM scheduler and ECA units to avoid a collision. If an action channel detects two simultaneous actions, one will be randomly delayed and the error status flag raised. The host system is responsible for clearing and recording the error, potentially for triggering post mortem.

ECA-6: *Scheduled events are irrevocable*

Once an incoming event has been received by an ECA unit, scheduled actions are irrevocable. If the DM may need to abort an incomng event, it should not broadcast the timing message until the incoming event can be committed.

ECA-7: *Timestamps are sequential*

Converting timestamp formats in hardware is expensive. The ECA unit requires a counter which increments without gaps. The proposed format is the number of 8ns cycles since 1970-01-01 TAI represented as a 64-bit unsigned value. Absolute timestamps from the DM and offsets in the condition table must use this format.

ECA-8: *Incoming event identifiers are fixed-length*

In order to facilitate matching incoming events with condition table entries in hardware, those must be identified by a fixed-length binary string. A multiple of 32-bit is preferred. If the LSA group insists on large identifiers, then those identifiers will need to be externally mapped to identifiers that can be used by the DM+ECA. The proposed format is a 64-bit value.

ECA-9: *Incoming Event Identifier matching pattern*

Sometimes, distinct incoming events generated by the DM may call for nearly identical handling handling by a FEC. In this case, it is helpful for condition table rules to match more than one incoming event identifier. Thus, the same condition rules can apply to a large class of incoming events. To differentiate between matching incoming events, the complete identifier is provided to RCs on the action channel outputs. If the LSA group choose to use an external mapping of a large identifier to a shorter hardware-friendly identifier, then they should take care that the bits are arranged such that it is still easier to categorize them for multi-identifier-matching condition rules.

ECA-10: *Condition table identifiers*

Multiple rules in the condition table can generate actions for a single channel (ECA-3). Some RCs may need to recognize which rule triggered the action. For example, a LEMO pulse generator must know whether the action was generated by an ON or an OFF event. To facilitate this, condition tables must include a customizable tag per rule. This tag is provided to RCs on action channel outputs.

ECA-11: *Wishbone event interface*

Receipt of incoming events must be possible via a single 32-bit Wishbone FIFO register. This permits Etherbone messages from the DM to utilize the compact FIFO mode message format. Furthermore, the register should be replicated to multiple addresses, facilitating sequential loading using system calls like memcpy() from the host system and/or embedded CPUs.

ECA-12: *Wishbone condition interface*

The condition table contents and registers must be accessible via a wishbone interface.

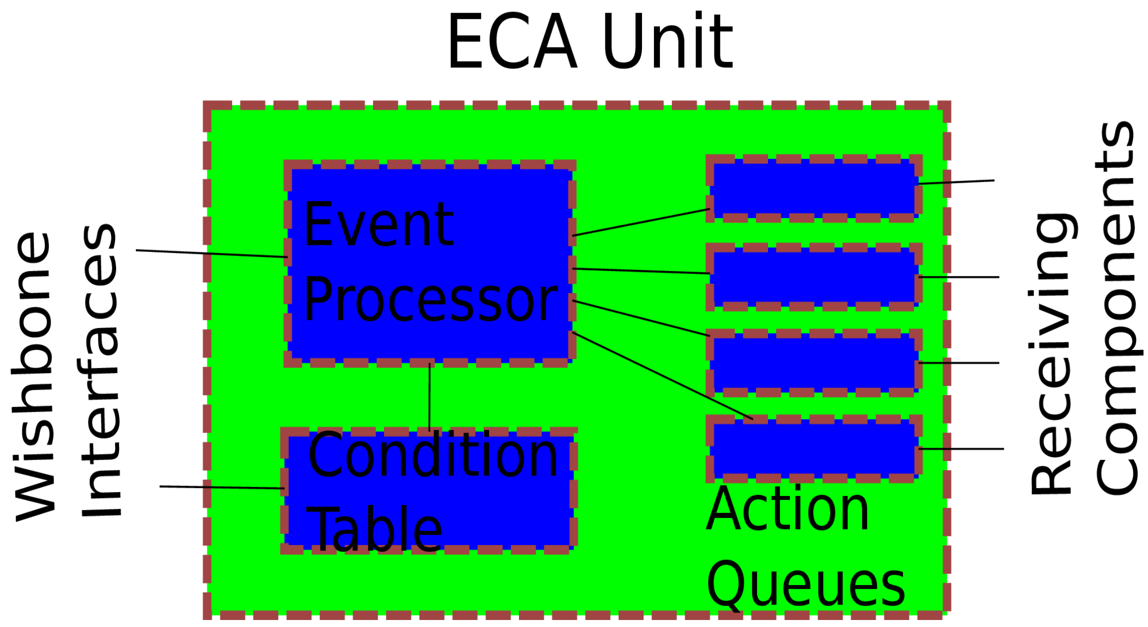
ECA-13: *Wishbone HW event bridge*

It is sometimes useful to receive incoming events from another source than the DM. To support this, there will be an optional component which translates externally generated events into a sequence of Wishbone operations suitable for delivery via ECA-11. This supports delivery of events from the host CPU and/or an embedded soft CPUs.

ECA-14: *Human-readable descriptions*

When inspecting an ECA unit, potentially over the network, it must be possible to easily identify what components are available for control. The ECA should include human-readable descriptions for itself and each connected action channel.

Proposed Architecture:



Format of incoming events

Incoming events must include: (id, abs_time, param).

id identifies the incoming event and is used to match the incoming event against rules in the condition table.
abs_time describes execution time of the event as a whole relative to Jan 1, 1970.
param is a user-defined parameter passed through the ECA to the RCs.

The length of *id* should be determined by the LSA group and the DM designer, preferably a multiple of 32-bits. The internal structure of the *id* is irrelevant to the ECA, but it should be chosen to make it easy for a prefix matching rule to select a useful group of incoming event identifiers.

The *abs_time* field must conform to the chosen timestamp format. Proposed as a 64-bit value interpreted as the number of 8ns cycles since Jan 1, 1970.

The *param* field is opaque to the ECA and forwarded end-to-end from incoming event to action output to an RC. Potential uses could be: a fine-grained phase-offset, target power level, etc.

The entire incoming event must be written as a sequence of 32-bit words, suitable for feeding into a FIFO register using the Etherbone protocol.

Format of condition table

Entries in the condition table include: (id, bits, channel, delay, tag)

id is matched against incoming events to determine if the rule is applied.
bits describes how many bits of the incoming event identifier are matched by *id.*
channel is the index of the action channel whose RC should take action when this incoming event arrives.
delay is an offset added to the event's *abs_time* to determine the absolute timestamp at which the action will be executed.
tag is a field of information passed “as-is” through to the RC when the rule is matched.

Format of action channels

The connection to RCs provides these signals: (strobe, id, tag, param, time)

strobe is 1 on the 125MHz cycle where an action should be taken. Otherwise it is 0.
id is the event id as received from the DM.
tag is the extra information entered in the matching condition rule for use by the RC.
param is the extra information that appeared in the received timing event
time is the execution time of the action (always current time for undelayed actions)

Design and implementation issues

The ECA unit is constructed as shown in the Figure. The Event Processor translates arriving incoming events into actions according to the rules condition table. These actions are then routed to the appropriate action queue. Finally, the action queues output the actions to their channels at the appropriate time. While simple in principle, each of these steps has limitations when subjected to very tight real-time requirements.

EP-1: *Condition table size*

The more conditions in the table, the harder it is to match them. The design so far has aimed for a ballpark of 256 conditions. If this is much smaller (say 16), then the Event Processor (EP) becomes trivial. If this is much larger (say 2000) then EP-4 becomes very important

EP-2: Incoming e*vent arrival rate*

Incoming events arrive at the ECA via the Wishbone interface. The required throughput is critically important to the implementation of the Event Processor (EP). If they can arrive 8ns after each other, then the EP must be capable of matching all the conditions in a single step. If there is 4us between them, it would be possible to use a single software-like process to iteratively compare each condition to each incoming event. Supposing incoming events (id, abs_time, param) are 5x32-bit in length, then delivery over Gigabit Ethernet takes 160ns. If there are additional incoming event sources, then the EP must be capable of greater throughput. The less time available, the more exotic the hardware needed. The current prototype can process one incoming event per 80ns for a condition table with 256 entries.

EP-3: *Multiple action results*

A single incoming event may match multiple conditions. This means that the EP may have to produce multiple action outputs for a single incoming event input. These outputs must all be delivered to the correct action queue, sometimes several actions per queue. The following restrictions apply,

EP-3.1: *Multiple action results per channel*

Without heroic design effort, a single action queue cannot accept more results than the number of cycles available to process a single incoming event. While it is possible to design a queue with multiple parallel inputs, the only approaches that I know of is a (prohibitively) expensive sorting network. For the 160ns example arrival rate, this sets a limit of 10 actions per channel per incoming event. In comparison to EP-3.2, this should probably be taken as a hard limit.

EP-3.2: *Total actions per incoming event*

If there are too many actions produced by one incoming event, then the interconnect between the EP and the queues must accommodate more than one action per cycle. The current EP prototype can only dispatch only one action per cycle, so to match the bandwidth in EP-2, a single incoming event should not generate more than 10 actions. It is to expand this using additional hardware.

EP-4: *Stall when rate exceeded*

The EP may stall the flow of incoming events when they exceed the maximum bandwidth supported (EP-2). This slows down the incoming rate without introducing losses.

EP-4: *Report error on congestion*

If the EP is overloaded and cannot accommodate incoming events, it must report Wishbone error rather than stall the source. If an action channel is full, the EP may be blocked indefinitely and it is unacceptable to potentially hang the entire Wishbone bus in this case.

CT-1: *Condition table changes are atomic*

It must be possible to reconfigure the condition table atomically. Changes should be written to temporary storage within the ECA. Enabling the new configuration must be atomic. Actions already queued for delivery to channels are unaffected. After switching configuration, all new events are processed using the new table in its entirety.

CT-2: *Match flexibility*

The matching power of condition rules affects the hardware cost. There were three candidates considered: equality, prefix-match, and bitmasked-match. Equality matching does not support ECA-9 and was rejected. Matching using a bitmask requires scanning the whole condition table, which impacts EP-2. To meet Gigabit Ethernet speeds, only 20 cycles are available. Thus, the EP would have to be replicated in hardware 13 times to support 256 condition table entries. This was deemed too costly and instead the current ECA prototype supports prefix-matching, akin to Internet routing table entries, which made it possible to search the table in log(n) time. If prefix-matching proves too inflexible for LSA group, then the condition table will need to be smaller.

CT-3: *Inspect configuration*

It must be possible to inspect both the contents of the active and inactive condition table.

AQ-1: *Sorted delivery order*

The action queues must output actions in sorted order. Unfortunately, even if the events arrived in timestamp sorted order, the delays added by the condition rules could break that ordering. Thus, the action queues must sort their actions.

AQ-2: *Queue depth*

Naturally, the more pending actions in the channel, the more memory they take and the harder it is to keep them sorted. Since each entry in the queue will be (at least) 150-bits long, it would be best to keep the majority of the queued actions in a RAM block. The current prototype recommends 256 actions per queue (the minimum for an Altera memory block).

AQ-3: *End-to-end latency*

There is a processing window from arrival of the incoming event to first possible output of an action. In the current prototype, this is in the ballpark of 512 cycles=4us (256 for EP and AQ). To clarify, this gap does not affect the timing accuracy of the action; it will still be output on the correct cycle. This window impacts how far in advance the incoming event must be delivered to the ECA unit in order for it to be processed in time. As always, it is possible to reduce this number at the cost of more exotic hardware.

AQ-4: *Overflow management*

If an action queue is overfilled, new actions cannot be added. This will almost certainly lead to lost actions, a very dangerous error condition. Thus, the ECA will report the current fill and maximum fill of each queue. The host system should monitor these values with a watchdog and report error if the queues ever exceed some safety threshold, perhaps 50% fill.

AQ-5: *Queue inspection*

For debugging, it must be possible to inspect the contents of the action queues. It is acceptable if the queue must be frozen / taken out of operation during this procedure.

Glossary and Abbreviations

- **timing message:** Message that is broadcast by the DM over the timing network. A timing message may contain one or more *incoming events*.
- **incoming event:** Input to the ECA unit. Examples of incoming events are
    - content of a timing message received from the DM,
    - request from the FEC host system or
    - digital input signal
    - an incoming event could also be a *timing event*, generated by (another) RC.

- **timing event:** Concept used at the front-end. It can be described as a trigger signal that is delivered by the timing system to a front-end. Example: Interrupt to front-end software.
- **AQ:** Action Queue – component of the ECA unit
- **CT:** Condition Table – component of the ECA unit
- **ECA:** Event-Condition-Action Unit - described in this document
- **EP:** (Incoming) Event Processor – component of the ECA unit
- **DM:** Data Master – component of timing system. It's tasks include timing message generation
- **FEC:** Front-End Equipment Controller
- **RC:** Receiving Component: Not a a component of the ECA unit but a component connected to a AQ
- **SCU:** Scalable Control Unit – standard FEC at FAIR
- **TR:** Timing Receiver, component of the timing system embedded in a FEC