TIMING AND SYNCHRONIZATION AT BEAM LINE EXPERIMENTS

Helena Blaettler Pruchova, Timo Korhonen, Paul Scherrer Institute, Villigen, Switzerland

Abstract

Some experiment concepts require a control system with the individual components working synchronously. At PSI the control system for X-Ray experiments is distributed in several VME crates, on several EPICS soft ioc servers and linux nodes, which need to be synchronized. A timing network using fibre optics, separated from standard network is used for distributing of time stamps and timing events. The synchronization of all control components and data acquisition systems has to be done automatically with sufficient accuracy and is done by event distribution. Data acquisition is synchronized by hardware triggers either produced by sequences in event generator or by motors in case of on-the-fly scans. Some detectors like EIGER with acquisition rate up to 22 kHz, fast BPMs connected to current measuring devices like picoammmeters with sampling frequences up to 26 kHz and photodiodes are integrated to measure beam properties and radiation exposures. The measured data are stored on various file servers situated within one BL subnetwork. In this paper we describe a concept for implementing such a system.

INTRODUCTION

The purpose of this work was to provide a user friendly method for synchronizing various detectors and devices when different devices are controlled by different IOCs. The beam line user needs a simple way of defining the pulse trains and sequences for the experiment consisting of various devices. Sequences could be runned very fast and it is easy for user to process it. Having a timing system that distributes events and time stamps enables to run aquisitions which are triggered by a unique master clock.

METHOD

Timing system of Swiss Light Source is based on Micro-Research Finland Oy timing hardware. The VME standard cards used mainly in our system are the event generator EVG-230 and event reciever EVR-230RF. Timing signals needed for synchronisation of subsystems are generated by the EVG and distributed to the receivers (see Fig. 1).

The receivers then in turn do the actions that belong to this event; the actions can be programmed by the user. The important functionality for this application is the possibility to create event sequences that are loaded to a sequencer RAM in the event generator. These sequences are in fact tables that define which event is sent out at which time, as defined by the clock of the event generator. This clock is typically (but not necessarily for this application) locked to the RF signal of the accelerator. The events in the table are sent out at times defined in the table by the hardware

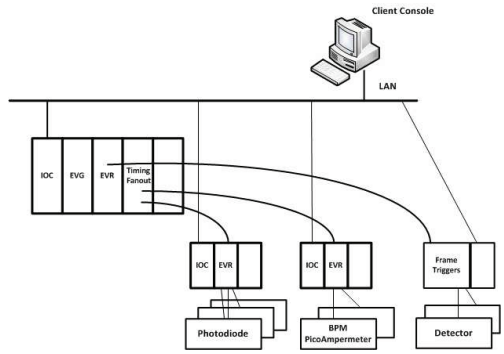

Figure 1: Hardware Structure of the Beamline.

after it receives a trigger. A trigger can be generated by software or via a hardware input or by a number of other methods that the hardware supports. More details are in the hardware documentation [1], [2]. An event sequence in the generator could look like the one shown in Fig. 2.

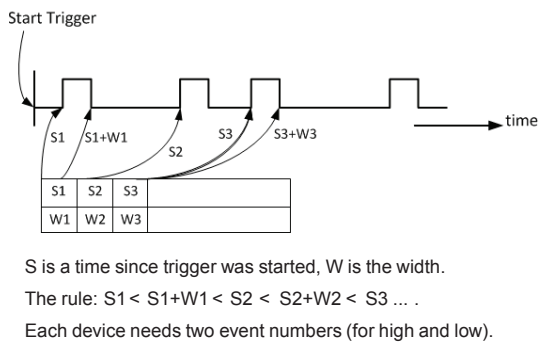

Figure 2: Creating of the sequence for one device.

The typically used EPICS support for the sequencers has been to use so called ”egevent” records, a single record that defines one event to be put into the sequencer. This approach does not lead itself to implementing an arbitrary sequence and for this reason we selected a different method.

To make it easy for the end user to define the trigger sequences, the user needs only to think about the time sequences for triggering the devices. The underlying system takes care of the tasks of allocating the event numbers and programming the sequences. The user starts by defining a timeline for each of the components in the experiment that need triggering. The timeline defines the synchronization pulses (start time, pulse width) relative to the start time (Figure 2). There can be an arbitrary number of these sequences, depending on the need of the experiment and limited only by the hardware capabilities, mainly by the number of available event codes.

ISBN 978-3-95450-139-7

Each trigger sequence needs two event codes, one for starting the action and one for ending the action. To program the timeline in the sequencer, the software puts a start event code in the sequencer, with the time defined by the start time in the table. It then puts an end event in the sequencer, with time defined by the start time plus the defined pulse width.

For the experiment one needs a number of trigger sequences like this. Handling just one is trivial but when there are several sequences the software has to sort and merge them before they are written in the hardware. One important constraint is to define a way to handle events that should occur simultaneously. The sequencer (and the EVG) can only send out one event at a time and the software has to handle this. Events from different sequences that have exactly the same time have to be handled in a way that they get put into different time slots. One collision causes a time shift of one sequencer clock cycle. The first implementation just ignores this but a more clever merge algorithm that combines the colliding event into one will be implemented in a later phase, after the first version is deployed.

SOFTWARE IMPLEMENTATION

The full implementation requires a method to program the sequencer, methods for the user to define the sequences and an application with a GUI for the end user.

The low-level implementation consists of two waveform records and one genSub record, with a user routine that writes the data in the waveform records to the sequencer. The user application needs to put the event numbers in one waveform and the times in the other one, and then tell the gensub record to process (see Fig. 3).

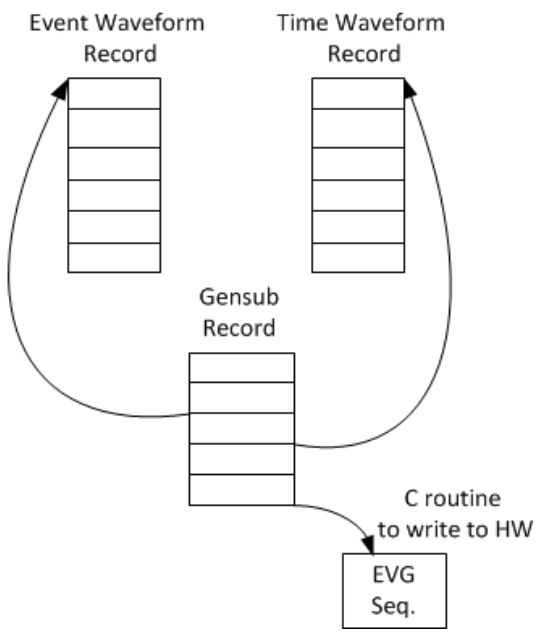

Figure 3: Epics records scheme.

The high-level implementation consists of one class that handles a collection of time sequences as described above, of one class to handle the event number reservation and a class to handle the individual time sequence. The class handling the event number reservations is rather simple, just having a pool of event numbers and the information if they have already been taken. When a new time sequence is created, the sequence just needs to reserve two event numbers using the class methods.

The class for the individual event sequences provides methods to create the timeline, with start and width definitions for the sequence. The collection class provides methods to manage (create, delete) the sequences and then deploy the whole collection to the hardware, using the EPICS channels as defined above.

A GUI can be based on any suitable toolkit that has access to the above classes. The first implementation is based on Matlab because it can be readily deployed on the beamline (see Fig. 4).

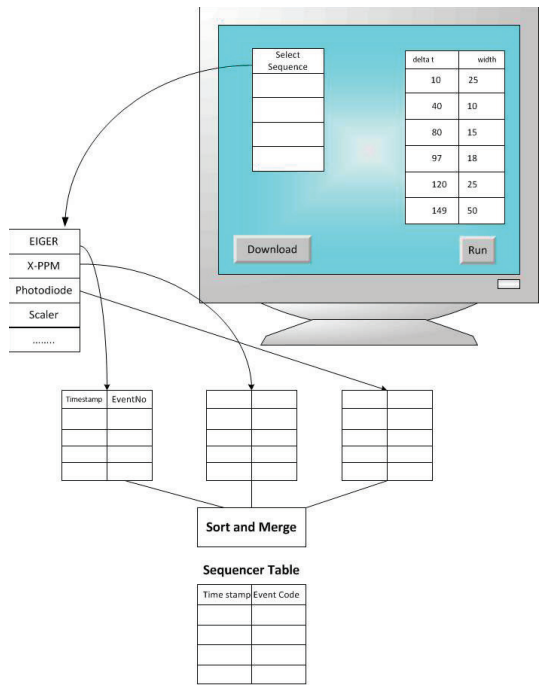

Figure 4: User Utility to create a final Sequencer Table.

The event receivers have also to be programmed correspondingly. Here the existing EPICS facilities are sufficient, but rather complicated for the user to handle. In the first stage this can be done by a supporting controls person, but the plan is to provide facilities to define which receivers should receive which sequences and to program them accordingly as a part of the normal workflow.

TEST RESULTS

To show how this system works in practice, we made a test setup in our laboratory, with one EVG, two event receivers and some other hardware that we trigger from the

ISBN 978-3-95450-139-7

EVRs. With this setup we can download a sequence definition to the EVG and run it. The oscilloscope screenshot in Figure 5 shows one example with four time sequences. The traces above show the whole sequence and the zooms in the lower part of the picture show zoom views of the individual triggers. Just as a simple proof we connected a signal from a signal generator to two ADCs that are triggered by two separate EVRs. Changing the sequences that trigger each of the ADCs we can verify that the signals are captured with the correct relative times.

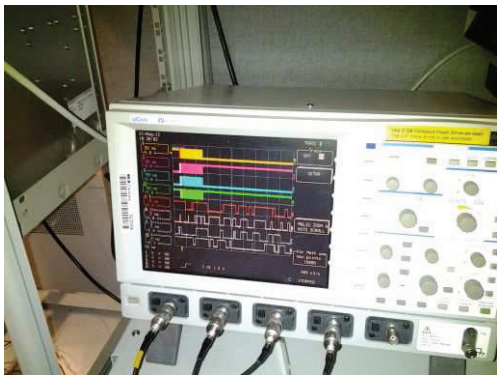

Figure 5: Pulse train taken with oscilloscope.

CONCLUSIONS

This method provides a user-friendly way for a beamline experimentator to define a complex time structure for an experiment where different elements and detectors can be triggered with different times, fully configurable by the user. The rather complicated manipulation of the low-level hardware can be hidden from the user by implementing the classes as described above, and the user can work just with the definition of the sequences and has not to think about the underlying event codes and their mapping.

This method was recently implemented at beam line for Coherent Small-Angle X-Ray Scattering where the single photon counting and fast framing X-Ray detector EIGER was commissioned.

REFERENCES

[1] http://www.mrf.fi/index.php/vme-products EVG-230TREF-001.Micro-Research Finland Oy. Vlitalontie 83 C, FI-00660 Helsinki, Finland.

[2] http://www.mrf.fi/index.php/vme-products EVR-230TREF-001.Micro-Research Finland Oy. Vlitalontie 83 C, FI-00660 Helsinki, Finland.

ISBN 978-3-95450-139-7