Contents

1 4kHz MCOR PS Control .......................... 1
   1.1 The Setpoint Table .......................... 2
   1.2 4kHz Readback History .......................... 2
   1.3 4kHz State Machine .......................... 2
       1.3.1 States .......................... 2
       1.3.2 Setpoint Table Processing .......................... 3
       1.3.3 Example .......................... 4

2 Spear Timestamps and Events .................. 4
   2.1 Timestamps .......................... 4
   2.2 Events .......................... 5

3 4kHz BPM History Buffers .................... 5
   3.1 History Timestamps .......................... 6
   3.2 Triggering the History Buffers .......................... 6
       3.2.1 Semantics of $\text{RARM}$ .......................... 6
   3.3 Examples .......................... 6

1 4kHz MCOR PS Control

Note: this memo doesn't describe all aspects of MCOR control but focusses on the fast control features. Other issues (ADC multiplexers, status monitoring, calibration...) are beyond the scope of this document. Also, this was put together very quickly and hence may not be free of errors...

Every MCOR channel features a state machine to manage a setpoint table for the DAC and 3 history buffers for the 3 on-board ADCs. (These history buffers record the same channel of the input MUX that is monitored by the slow readbacks).
1.1 The Setpoint Table

The setpoint table (<corr>:CurrSetpt) can hold up to a maximum of 2000 values [can be increased but you’ll need EPICS-3.14 to write large arrays]. It is possible to write less values than the maximum - EPICS will automatically know the number of elements that have been written to the setpoint table (this number is itself a read-only PV: <corr>:CurrSetpt.NORD). In particular, it is possible to write a single value to the table. The setpoint table supports Spear Timestamps (<corr>:CurrSetpt.TSE==-2) which means that it is possible to determine exactly when the first (and therefore all subsequent) setpoint was written out.

1.2 4kHz Readback History

The PVs: <corr>:Curr1MuxADCTbl, <corr>:Curr2MuxADCTbl and <corr>:VoltMuxADCTbl hold 4kHz history waveforms of the readback ADCs. History data acquisition is started at the same 4kHz cycle the first setpoint is written to the DAC and it stops when the buffer is full. The ADC waveforms also support Spear timestamps. Because the state machine always starts the setpoint table and the readback waveforms at the same clock cycle, the Spear Timestamp attached to CurrSetpt and xxxMuxADCTbl should always be the same.

Note that whereas there is an interpolation feature (see below) for the setpoint tables, the sampling rate of the ADCs is always fixed at ≈ 4kHz (Frev/320). Also, unlike the setpoint table, the readbacks do not loop/wrap around. Once the buffers are full, data acquisition stops.

1.3 4kHz State Machine

The state machine is responsible for computing DAC setpoints (among other things, this involves scaling according to the calibration, which is a topic not discussed here) and for processing ADC readbacks (writing history buffers and running decimation filters for slow readbacks).

The state machine is always driven by the 4kHz (Frev/320) timestamp / clock facility (sidenote: should this clock signal be lost, the software tries to switch to an internal 4kHz clock mode – global synchronization will be lost, however).

1.3.1 States

The current state of the state machine is reflected by the ControlState PV. this PV is at all times writable by the user. Under certain circumstances, the state machine performs state transitions itself. The following states are defined:
HALT all processing is halted. New setpoint table can be written. The state machine must be explicitly brought out of HALT state by the user.

ARM processing is halted. New setpoint table can be written but this automatically triggers a transition into ‘RUN’.

REMOTE setpoint table is ignored. Setpoints are accepted from the dedicated fast ethernet (fast orbit feedback).

RUN the state machine is processing the tables. No new setpoints can be written in this state. The only PVs that can be modified in ‘RUN’ state are ControlState, CurrInterSteps and CurrScale. LoopIter can be changed but this only will take effect on the next run.

After processing the setpoint table, the state machine returns to ‘ARM’. Processing may be aborted at any time by changing ‘ControlState’.

EVENTx Similar to ‘RUN’. Processing is not started immediately, however, but only at the arrival of a synchronization event (Spear Event).

1.3.2 Setpoint Table Processing

At every 4kHz cycle, starting when entering the ‘RUN’ state (or at the reception of an event if in one of the ‘EVENTx’ states a new setpoint is written to the DAC. Some details are controlled by the following PVs:

**Interpolation Steps** (\(<corr>\):CurrInterSteps – default: 0)

If so desired, any number of interpolation steps can be inserted between two setpoint table values. The processor performs linear interpolation between the table values or between the current value and the 1st table value (when processing starts).

Example: current value is 2A. Two setpoints [5,10] are written and interpolation steps are ‘1’. Hence, the processor will write the sequence: 3.5, 5, 7.5, 10.

If the parameter is 0, no interpolation is performed, i.e., setpoint table values are written as-is.

**Scale** (\(<corr>\):CurrScale – default: 1)

The every sample written to the DAC is scaled by this value. This makes it possible to change the amplitude of a waveform without having to generate a new table.

**Loop iterations** (\(<corr>\):LoopIter – default: 1)

This parameter controls how many times the table is to be processed. I.e., after the last value has been written, processing is
resumed from the beginning of the table for LoopIter - 1 times. Any number less than 1 instructs the processor to continue indefinitely. Processing needs to be stopped by switching into one of the ‘HALT’ or ‘ARM’ (or ‘REMOTE’) states.

Note that it is legal to modify CurrScale and CurrInterSteps on the fly, while processing is ongoing thus allowing for “knobbing” the amplitude/frequency of a continuous waveform.

While LoopIter can be modified after processing has started, the modification will only take effect the next time processing is started.

### 1.3.3 Example

Load a corrector with a setpoint table, arm BPMs to start buffering history 1s after the corrector starts:

```plaintext
// make sure table doesn’t start
// when written.
lcaPut('mycorr:ControlState','HALT')

// load table. If not 'HALT'ed,
// this would trigger processing
lcaPut('mycorr:CurrSetpt',mytbl)

// loop forever
lcaPut('mycorr:LoopIter', -1)

// wait for event 1
lcaPut('mycorr:ControlState','EVENT1')

// post-trigger 1s, wait for event 1
lcaPut('mybpm:history.RARM',-4000+1)

// fire
lcaPut('Spear:Event1',1)
```

## 2 Spear Timestamps and Events

### 2.1 Timestamps

The Spear Timestamp is a 64-bit counter incremented at each 4kHz clock cycle ($F_{rev}/320$). This counter is implemented in the “Timestamp Master IOC” and is reset to zero either manually or when the master IOC is powered up (but not when it is reset).

Spear Timestamp information is distributed on a wire and is available to other IOCs provided that they are equipped with a time-stamp
VME module (aka. TSSM). The most important IOCs with timestamp capability are the MCORs and the BPMs (including photon BPMs).

Some (but not all) PVs on a timestamp-capable IOC are tagged with a Spear Timestamp in addition to the ordinary EPICS timestamp (wall-clock time) which is associated with every PV and which can be retrieved along with the PV value over channel access.

The EPICS timestamp is in “unix-style”, i.e., it consists of a ‘seconds’ and a ‘nanoseconds’ part. [LABCA passes EPICS timestamps as complex numbers with the seconds and nanoseconds in the real and imaginary part, respectively].

PVs that are tagged with a Spear timestamp carry the spear timestamp MODULO 1E9 in the nanoseconds part.

You can find out if a PV is tagged in this way by looking at the TSE field of the EPICS record the target PV is a field of (just append ".TSE" to the PV name). Tagged records have a TSE value of \(-2\).

In addition to timestamping selected PVs, an IOC equipped with a TSSM (timestamp + sync module) also provides the running 64-bit timestamp as a dedicated PV: `<ioc>:SpearTimestamp`.

### 2.2 Events

The timestamp hardware is also capable of distributing 7 different, global events to all IOCs equipped with a TSSM. Events can be sent on each 4kHz clock cycle. They can be used to trigger e.g., corrector setpoint tables or BPM history snapshots or for other synchronization purposes.

There are seven binary PVs, Spear:Event1 .. Spear:Event7 which, when set to 1 will result in broadcasting the corresponding event.

I suggest that we reserve Event7 for the orbit interlock, and Event6 for development..

The event PVs are automatically reset to 0 after a period of about 100ms.

### 3 4kHz BPM History Buffers

The BPM processing system features a history buffer where orbits are stored at a rate of 4kHz. The currently (hardcoded) buffer depth is 4000 samples/orbits.

Since BPM processing is partitioned into two IOCs in B116 and B132, there are two buffers which must be synchronized e.g. by using Spear Events for triggering or by aligning their Spear Timestamps. The respective PVs are `116-BPM:history` and `132-BPM:history`. 
3.1 History Timestamps

The history PVs are tagged with a Spear Timestamp. The timestamp always refers to the first/oldest element in the history (no matter if pre- or post-triggering is used – see below).

3.2 Triggering the History Buffers

The history PVs have two states: ‘armed’ or ‘idle’. In ‘armed’ state the history waits for a triggering event and upon reception, starts writing data into the buffers. Once in ‘armed’ state, the history is ‘busy’ and cannot be accessed. Most notably: at the moment it cannot be disarmed.

After filling up, an ‘armed’ buffer returns to ‘idle’ state and can be read and rearmed.

A history buffer is armed by setting its RARM field (The respective PVs are 116-BPM:history.RARM, 132-BPM:history.RARM) to a non-zero value:

3.2.1 Semantics of RARM

The number stored in RARM is the number of “pre-trigger” samples to be stored in the buffer. e.g., if RARM=4000, the buffer “looks into the past”, e.g. stores 4000 samples preceding the trigger. It is possible set RARM to a negativ number thus giving post-trigger capability. RARM=-8 skips 8 samples after trigger before putting data into the buffer. in this example, the history’s resulting Spear timestamp would differ by 8 from “trigger-time”.

The number of pre-trigger samples always must be a multiple of 8, since the least significant 3 bits of the RARM PV are used to specify the triggering method. Values of 1..7 indicate that the history is to be armed on the respective Spear Event, a value of zero means “soft triggering”, i.e. writing the RARM field is the trigger. Since RARM must always be non-zero, it is not possible to use 0 pre-trigger samples together with software triggering.

3.3 Examples

RARM = 1: 0 pre-trigger samples, wait for Spear Event 1 to trigger.
RARM = 8006: pre-trigger == full buffer depth (any number > buffer depth is clipped), wait for Spear Event 6.
RARM = -16: post-trigger 16 samples using soft-trigger.

Synchronously trigger two histories, zero pre-trigger samples:

// arm on event 2
lcaPut('116-BPM:history.RARM',2)
lcaPut('132-BPM:history.RARM',2)

// raise event
lcaPut('Spear:Event2', 1)