CNPEM
Centro Nacional de Pesquisa
em Energia e Materiais

Laboratório Nacional
de Luz Síncrotron

IOC User Guide
Version 1.0
February/2017

Beam Diagnostics Group (DIG)
Brazilian Synchrotron Light Laboratory (LNLS)
Brazilian Center for Research in Energy and Materials (CNPEM)

About this manual

This guide provides information and instructions about the setup and operation of the Timing System IOC. Information about the timing system hardware, or firmware can be found in the corresponding manuals.

Contents

1 Overview

The Timing System IOC is responsible for managing and providing process variables (PVs) for control of the Timing System modules. The Timing System functions are divided into categories and presented along with the process variables needed for configuration in section Timing System Operation. Further detail about the PVs is presented in section Timing System Process Variables.

2 IOC Setup

This section describes the procedure for correctly installing and configuring the IOC for use.

2.1 Installation

The Timing System IOC requires EPICS base 3.14, synApps 5.8, and StreamDevice 2-7-7 installed. Make sure to provide their intallation paths to the IOC (RELEASE file) before building the IOC.

2.2 Editing the RELEASE file

The RELEASE file, located at /<IOC directory>/configure/RELEASE, contains the paths for external applications. Generally, this is the only file that requires modification to build the IOC in a new machine. Make sure that the paths specified in the file agree with the applications installation locations.

2.3 Building the IOC

In order to build the IOC, change the working directory to the IOC top directory and run make. Clean the directory when rebuilding the IOC.

```
make clean uninstall install
```

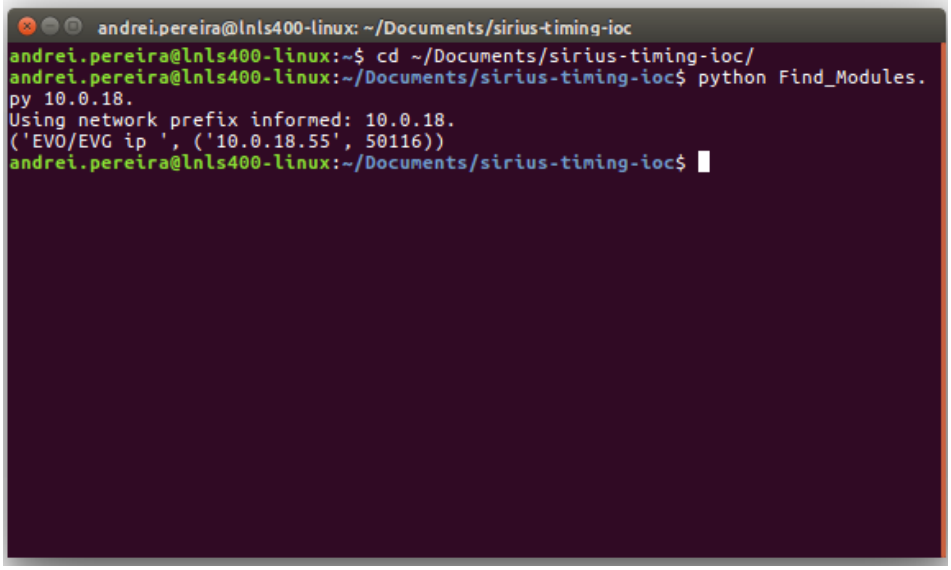
2.4 Finding the modules in the network

The Timing System modules are DHCP by default. Therefore, the next step to set up the IOC is to find each module's IP. The top directory of the Timing System IOC contains a python script for finding the Timing modules IPs called *Find_Modules.py*. The script searches for the modules using the provided subnetwork prefix. In order to run it, it is necessary to have python 2 installed. From the top IOC directory, the following shell command can be used:

```
python Find_Modules.py <network prefix>
```

The ouput is the list of connected modules. Each line shows the info for a module, that is, its type and configuration (i.e., EVO/EVG, EVO/EVR, EVO/FOUT, or EVE), followed by its IP and port. The example of figure 1 shows the output of *Find_Modules* for a network with only an EVG module with IP 10.0.18.55, and port 50116.

Figure 1: Find_Modules.py output



The port number is used for identifying the devices, since each Timing System module port number is unique. Table 1 presents the code and port number associated with each of the Timing System modules.

The modules' IP and port number are used for configuring the UDP/IP communication between the IOC and Timing System modules in the startup script. This procedure is described in Editing the startup script.

Table 1: Timing System modules Port Number

Module	**Code**	**Port**
EVO 1	P1916001	50111
EVO 2	P1916002	50112
EVO 3	P1916003	50113
EVO 4	P1916004	50114
EVO 5	P1916005	50115
EVO 6	P1916006	50116
EVO 7	P1916007	50117
EVE 1	P1816002	50121
EVE 2	P1816003	50122
EVE 3	P1816004	50123
EVE 4	P1816005	50124

2.5 Editing the startup script

The startup script should only be modified when installing the IOC in a new subnetwork, after a modification to the Timing System structure, or when a modification to the IOC initialization is desired.

The script path is *<ioc directory>/iocBoot/ioctiming/st.cmd*.

The first step when configuring the IOC in a new network is to configure the IP ports for each device that will communicate with the IOC. This is accomplished by the command *drvAsynIPPortConfigure* in the startup script. For instance, suppose we have the following list of IP port configuration commands:

```
drvAsynIPPortConfigure ("EVG1", "10.0.18.55:50116:50116 udp",0,0,0)
drvAsynIPPortConfigure ("EVR1", "10.0.18.111:50111:50111 udp",0,0,0)
drvAsynIPPortConfigure ("EVR2", "10.0.18.112:50112:50112 udp",0,0,0)
drvAsynIPPortConfigure ("EVE1", "10.0.18.58:50121:50121 udp",0,0,0)
drvAsynIPPortConfigure ("EVE2", "10.0.18.54:50122:50122 udp",0,0,0)
drvAsynIPPortConfigure ("EVE3", "10.0.18.123:50123:50123 udp",0,0,0)
```

The first command defines a port called *EVG1* with IP 10.0.18.55, port number 50116, local port number 50116, and UDP protocol.

IPs defined in the script that are no longer valid should be updated before running the IOC. IP port configuration commands can be added or commented out, depending on the number of modules used.

Further detail about configuring the IP ports is provided in section 5.1.

2.6 Running the IOC

In order to run the IOC, change the working directory to *iocBoot/ioctiming*. Modify the *st.cmd* script permission so that it can be executed:

```
chmod +x st.cmd
```

Execute the startup script:

```
./st.cmd
```

The IOC should start running and some messages reporting the startup status should be printed in the IOC shell. IOC misconfiguration will most likely cause error messages to be displayed during the startup procedure.

3 Timing System Operation

3.1 Initialization

On IOC startup, the databases and sequencer programs are initialized. The autosave tool looks for save files to restore the IOC records' fields as they were in the last time that the IOC ran. The sequencer programs "EVGSetup" and "EVRESetup" check the configuration status of the timing modules. For modules that are disconfigured, a download operation is performed, transferring the select and set point parameters defined in the IOC to the module. For modules that are already configured, an upload operation is performed, transferring the module parameters to the IOC status and read-back value PVs.

The messages printed by the IOC shell are related to database, sequencer programs, and autosave initialization. Common errors and solutions are described in section Common Errors and Causes.

3.2 Download and Upload of parameters

It is possible to download or upload parameters to/from the timing modules at any time after the IOC has initialized. This is useful when connecting the IOC to a module that is already running. If the module is configured differently from the desired and the IOC PVs contain the right settings, it is possible to tranfer the settings all at once by setting the *download* PV. On the other hand, if the IOC status parameters or read-back values are not updated for some reason, setting the *upload_T* PV will update all the status and RBV PVs with the values in hardware.

3.3 First configuration steps

The first configuration steps for the timing system consist of configuring the modules inputs settings. The configuration varies from module to module, since each has its own set of inputs.

3.3.1 EVG

The first step for configuring the STD-EVO as EVG is to assure that the module function has been correctly configured by the EVG Setup sequencer program on IOC startup. The funselRBV PV displays the module configured function. In the case that the module is incorrectly configured, it is possibble to modify the module configuration by setting the appropriate value to the funsel PV.

As soon as the module receives a configuration, i.e., EVG, EVR, or FOUT, the alive counter starts incrementing. Although the alive couter info may be useful, a better aproach for checking whether the device is configured is to verify the statdev PV, which displays the device status as *WAITING* when the module has not yet been configured, or *RUNNING* otherwise.

The network connection status is displayed by the network PV.

Another important step when configuring the EVG is to guarantee that both the RF clock and the *event clock* are within limits. The *RFIN* (RF input) must receive a signal within the range of 60 - 540MHz. The *event clock* is the RF frequency divided by the *RF divider*, and must be within the range of 60 - 135MHz. The rfdiv PV sets the *RF divider*, defining the *event clock* frequency. The rfstat PV indicates the *event clock* status.

After configuring the RF input, the *ACIN* must be configured. The *ACIN* input receives the mains signal and has an associated *AC divider*. The output of the *AC divider* is used for triggering the EVG event distribution. Therefore, the divider should be set according to the desired triggering frequency, using the acdiv PV. It is also necessary to enable the AC input in order for the AC trigger to be generated. The *AC Line* enable/disable is done by setting the acen PV.

The EVG module does not output event codes (besides the idle event) or *Distributed Bus* clocks until the module is enabled. Enable/disable of the module is done through the enable PV.

3.3.2 EVR/EVE

The first step for configuring the STD-EVO as EVR is to assure that the module function has been correctly configured by the EVRE Setup sequencer program on IOC startup. The funselRBV PV displays the module configured function. The STD-EVE module (EVE) is always configured as *Event Receiver*, not requiring checking. In the case that the STD-EVO is incorrectly configured, it is possible to modify its configuration by setting the funsel PV accordingly.

As soon as the module receives a configuration, i.e., EVG, EVR, or FOUT, the alive counter starts incrementing. Although the alive counter info may be useful, a better aproach for checking whether the device is configured is to verify the statdev PV, which displays the device status as *WAITING* when the module has not yet been configured, or *RUNNING* otherwise.

The network connection status is displayed by the network PV.

The next step when setting the module up is to check whether the *UPLINK* link is established, i.e., the module is receiving the *Event Generator* messages. The *UPLINK* connection status is displayed by the link PV. In order for the EVR or EVE to correctly receive the *UPLINK* information, it is necessary that the module's clkmode PV is correctly configured according to the *UPLINK* signal frequency.

Another important variable is the interlock input status, which is displayed by the inhs PV. The interlock input only affects the module's outputs whose interlock function is enabled.

The EVR and EVE modules do not output any trigger or clock until the module is enabled. Enable/disable of the module is done through the enable PV.

3.4 Saving the IOC parameters

The save function saves all select and set point PVs so that the user is able to use the download command to configure the module in a single step. In order to save the IOC current settings, it is necessary to set the corresponding module's SAVE PV to 1.

3.5 Event Configuration

All the event PVs use the same name prefix as the EVG, since these PVs affect the EVG's event generation.

The events generated by the EVG are configured from a fixed length set of events loaded in the IOC initialization. There are four parameters which need to be configured for each of the events.

The event code is determined by the EVCODE{idx} PV, where {idx} corresponds to the index of the event being configured. If an event is not going to be used, it must be set INACTIVE by setting the respective event code value to 0 (zero).

The time interval between the *Event Sequencer* trigger and the event transmission is determined by the event timestamp, which is set using the EVTIME{idx} PV. The timestamp is set in *event clock* period units.

The transmission parameter determines whether the event is transmitted periodically at every *Event Sequencer* trigger, or only when an injection is performed. The transmission parameter is set using the EVTRANSM{idx} PV.

The event mode determines whether the event has a fixed timestamp, or the timestamp increments depending on the bucket that is receiving the injection. The mode selection is done by setting the EVMODE{idx} PV.

The maximum number of events is defined in the IOC initialization. The addition of more events is done by editing the IOC startup script (see section Event instantiation).

3.6 Event Generation

All the event generation PVs use the same name prefix as the EVG, since these PVs affect the EVG's event generation.

The event generation is controlled by the Injection state machine. The STATEMACHINE PV displays the state machine's state. When the state machine is in the *Stopped* state, no event is being transmitted by the EVG. Setting the RUNSEQ PV to 1 causes the Injection state machine to prepare for running. The state machine switches to the *Preparing to Run* state, while the EVG receives the events to be transmitted. If the EVG has successfuly received the events, the state machine goes to the *Running* state. In this state, the EVG transmits the events configured with mode *ALWAYS*, i.e., events that need to be transmitted constantly. Only when the state machine is in the *Running* state it is possible to trigger an injection.

In order to inject it is first necessary to define the list of buckets that are going to receive an injection. The bucket list is defined by the ITBL PV.

If the bucket list is defined, the *AC Line* and EVG are enabled, and the Injection state machine is in the *Running* state, the injection is ready to be triggered. The injection command consists of writing 1 to the INJSEQ PV. After an injection command, the state machine switches to the *Preparing to Inject* state and writes the injection events to the EVG. If the write operation succeeds, the EVG starts injecting and the state machine switches to the *Injecting* state. After all the listed buckets have received an injection, the state machine returns to the *Running* state.

The injection can be stopped by setting the STOPINJSEQ PV. This command returns the state machine to the *Running* state.

The state machine can be stopped at any moment by setting the STOPSEQ PV to 1. This command returns the Injection state machine to the *Stopped* state.

If the events' configurations are modified while the Injection state machine is *Running*, the next injection will already use the new event configuration, but the events transmitted during the *Running* state will remain unchanged. In order to modify both the events transmitted during the injection and always, it is necessary to restart the Injection state machine. This can be done in two different ways: by setting the RESTSEQ PV while in *Running* state, or by stopping the Injection state machine and running it again.

3.7 Trigger configuration

The modules responsible for the trigger generation in the Timing System are the *Event Receivers*. The EVR and EVE modules monitor some configured events, that, when received, generate a trigger in the configured outputs.

There are 16 OTP channels for monitoring events in each *Event Receiver*. The definition of a channel monitored event is done by setting the event{idx} PV of the receiver, where {idx} is the channel index. The delay, width, and polarity of the output trigger are configured through the delay{idx}, width{idx}, and pol{idx} PVs, respectively. The width and delay are both specified in *event clock* period units. It is also possible to specify the number of pulses generated by an OTP channel when the monitored event is received by setting pulses{idx}. If the number of pulses specified is greater than 1, the output is a pulse train with duty cycle 0.5, period of 2 $\times$ trigger width, and the first pulse delay is the specified for the trigger.

In the case of the STD-EVO/EVR, the OTP0 - OTP11 outputs are directly mapped to the rear panel plastic optical fiber outputs OTP0 - OTP11.

Both STD-EVO/EVR and EVE have 8 configurable outputs in the front panel (OUT0 - OUT7). Although the EVR front panel outputs are SFP with LC Duplex connectors, while the EVE has electrical outputs, both modules have the same output configuration options. Each OUT output has a sel{idx} PV associated with it, which selects the signal to be output by OUT{idx}. The signal can be any of the *OTP* channels outputs or *Distributed Bus* clocks.

When the source selected is an OTP channel, finer delays can be applied in addition to the OTP delays. The *RF delay*, rfdly PV, provides an $\frac{event\ clock}{20}$ resolution delay that can range from 0 to 19 (steps of $\frac{1}{20}$ *event clock*). When rfdly is set to 31, the RF delay is updated by special events that the EVG can be configured to transmit. The event codes 0x40 - 0x53 set the *Event Receiver*'s RF delay to 0 - 19, respectively.

The finest OUT delay is the fine delay, which has resolution of 5 ps. The fine delay is set by the finedly PV.

The OUT channels also provide an interlock function. When the itl{idx} PV is set to 1, the interlock function is enabled. Whenever the *Event Receiver* interlock input status is asserted, and the OUT{idx} interlock function is enabled, OUT{idx} is inhibited.

3.8 Timestamping function

The timestamping function of the Timing System is configured in many levels. First, all the modules need to be synchronized with the same timestamp. In order for that to happen, the first step is to configure the EVG timestamp settings.

The EVG UTC timestamp is set using the UTC PV. Modifying the UTC timestamp causes the EVG to broadcast the UTC timestamp to the Timing System. By default, the EVG UTC timestamp is initialized by the *EVGSetup* sequencer program with the linux UTC time. The EVG subsecond timestamp is incremented by the *event clock* and reset whenever the EVG receives a PPS signal. The subsecondRBV PV displays the timestamp subsecond count. The PPS (Pulse-per-second) is the signal responsible for updating the UTC timestamp. The source for the PPS can be configured using the TIMESRC PV. Modifying the PPS source causes the EVG to broadcast the UTC timestamp to the Timing System. Whenever desired, the UTC timestamp broadcast can be forced by writing 0 (zero) to the UTC PV. The PV value is not modified by a 0 write.

After the EVG timestamp settings are configured, the *Event Generator* is able to update the Timing System timestamps and the next step is to configure the *Event Receivers* timestamp options.

The EVR and EVE UTC timestamps are displayed by the UTCRBV PV. The subsecond counter value can be read from the subsecondRBV PV. The UTC timestamp of the event receivers is set when the EVG executes a timestamp broadcast and updated by a PPS. The PPS source for the EVR or EVE is configured by the TIMESRC PV.

After the EVR and EVE timestamp options are configured and the modules are synchronized with the EVG, it is desirable to configure the *Event Receivers* to store the timestamp in which monitored events are received.

Each EVR or EVE OTP channel has a parameter, represented by the TIME{idx} PV, which is used for enabling the timestamping function for OTP{idx}. When the timestamping function of OTP{idx} is enabled and the event being monitored (event{idx}) is received, the event code and the module's timestamp (UTC and subsecond) are stored in the *Timestamp FIFO*.

The *Timestamp FIFO*, also referred to as *Timestamp log*, is responsible for storing event-timestamp sets, which are periodicaly read by the IOC. The LOGCOUNT PV indicates how many event-timestamp sets are stored in the hardware's *Timestamp FIFO*. The *Timestamp log* information is pulled from the hardware and stored in three different buffers of the IOC: UTCbuffer, SUBSECbuffer, and EVENTbuffer, which hold, respectively, the list of UTC, subsecond, and event values. The LOGSOFTCNT PV is useful to keep track of the number of new events in the buffer. This variable is incremented whenever a set of event-timestamp is pulled from the *Timestamp FIFO*. When the LOGSOFTCNT reaches the maximum positive integer value it has an overflow, switching to the smallest negative integer value. The STOPLOG PV stops the *Event Receiver* timestamping function, i.e., while set, the STOPLOG PV prevents the module from storing new timestamps in the *Timestamp FIFO*. The RSTLOG PV resets the *Timestamp FIFO*, clearing it from all the stored values.

If the EVR or EVE module loses the *UPLINK* signal, the *Timestamp FIFO* is stopped and remains in this state until a reset command is issued, followed by the disabling of the STOPLOG PV.

3.9 Clock distribution

The *Distributed Bus* is a feature of the Timing System that allows for the distribution of 8 different clocks simultaneously. In order to use the *Distributed Bus*, it is necessary to configure the EVG to output the desired clocks, and also, the *Event Receivers* to transmit the clocks to the accelerator devices.

The EVG configuration consists of enabling the relevant *MUX* channels and setting the corresponding *MUX* dividers. The *MUX*{*idx*} is responsible for generating the *DBUS*{*idx*} clock. The *MUX* channels input is the *event clock*, which is divided by the corresponding *MUX* divider in order to generate the desired *DBUS* clock. The muxen{idx} PV enables the *MUX*{*idx*} channel. The muxdiv{idx} PV sets the *MUX*{*idx*} divider value. The output of the *MUX*{*idx*} divider is the *DBUS*{*idx*}.

After configuring the EVG, the next step is to configure the EVR and EVE modules. In order to output a given *DBUS* clock in the *Event Receiver* front panel, it is necessary to set the sel{idx} PV (*OUT*{*idx*} source selection) with the appropriate option. The *DBUS* clocks are available by setting the PV with the strings "Dbus0" - "Dbus7".

3.10 Providing clock multiple of event clock (EVE)

The EVE module is capable of outputting a square waveform with frequency multiple of the *event clock*. The clock output is configured by the cmsel PV.

4 Timing System Process Variables

4.1 EVG

4.1.1 STD-EVO Configuration

PV name	Description	Data Type
alive	Alive counter	ulong (32-bit)
funsel	STD-EVO configuration	enum [16: "FOUT"][17: "EVR"][18: "EVG"][32: "EVE"]
funselRBV	"funselRBV" PV Read Back Value	enum [16: "FOUT"][17: "EVR"][18: "EVG"][32: "EVE"]
rfdiv	Prescaler for RF clock divider	long (min=1, max=15)
rfdivRBV	"rfdiv" PV Read Back Value	long (min=1, max=15)

The *Configuration* parameters define the STD-EVO module basic functionality. The *funsel* parameter is used to configure the module as EVG, EVR, or FOUT.

The *alive* counter works as a heartbeat, and starts to count as soon as the module receives one of the possible configuration options: EVG, EVR, or FOUT. The *EVG Setup state machine* uses the *alive* information to determine the status of the network connection.

On startup, the *EVG Setup state machine* will set the *funsel* parameter in the case of the module being disconfigured.

The *rfdiv* parameter defines the RF frequency divider to be used. The output of the RF divider is the *event clock*.

4.1.2 EVG Control and Status

PV name	Description	Data Type
enable	Enable/Disable EVG	bool [0: "OFF"][1: "ON"]
enableRBV	"enable" PV Read Back Value	bool [0: "OFF"][1: "ON"]
seqen	Enable/Disable Event Sequencer	bool [0: "OFF"][1: "ON"]
seqenRBV	"seqen" PV Read Back Value	bool [0: "OFF"][1: "ON"]
rfstat	RF Input Status	bool [0: "loss or out of range"][1: "normal"]
seqstat	Event Sequencer Status	enum [0: "Stopped"][1: "SeqRAM 1"][2: "SeqRAM 2"]

The *enable* parameter enables/disables the module. When the module is disabled, it does not output any event or distributed bus clock.

The *seqen* parameter enables/disables the Event Sequencer.

The *rfstat* PV indicates the RF input status. The RF status will be *0* if either the signal was lost or is out of range. Setting the RF divider incorrectly may cause the signal to go out of range. The *seqstat* PV displays the current status of the Event Sequencer.

seqen is used by the *Injection state machine* to control the injection procedure and MAY NOT BE SET MANUALLY during normal operation.

> If *rfstat* goes to *0* (RF signal is lost or out of range), after restoring the RF signal, the *rfdiv* parameter MUST be set again (even if the value does not change) in order for the status to return to normal.

4.1.3 AC Line

PV name	Description	Data Type
acen	Enable/Disable AC line input	bool [0: "OFF"][1: "ON"]
acenRBV	"acen" PV Read Back Value	bool [0: "OFF"][1: "ON"]
acdiv	Prescaler setting in AC divider	long (min=1, max=1073741824)
acdivRBV	"acdiv" PV Read Back Value	long (min=0, max=1073741824)

The *acen* parameter enables/disables the *AC Line* (*ACIN* input of EVG). The *acdiv* parameter defines the AC divider value. The output of the AC divider defines the rate of the *Event Sequencer* trigger.

Example Setting the AC divider value to 30, for a mains signal of 60Hz, results in a 2Hz pulse. Consequently, the *Event Sequencer* is triggered every 2 seconds, while *SeqRAM 1* or *SeqRAM 2* is running.

4.1.4 MUX{idx} {idx}=0...7

PV name	Description	Data Type
muxen{idx}	Enable/Disable MUX{idx}	bool [0: "OFF"][1: "ON"]
muxen{idx}RBV	"muxen{idx}" PV Read Back Value	bool [0: "OFF"][1: "ON"]
muxdiv{idx}	Prescaler setting in MUX{idx} divider	long (min=1, max=1073741824)
muxdiv{idx}RBV	"muxdiv{idx}" PV Read Back Value	long (min=0, max=1073741824)

The *muxen*{*idx*} parameter enables/disables the corresponding MUX{idx} channel in the *Distributed Bus* (*DBUS*). The *muxdiv*{*idx*} parameter sets the MUX{idx} divider. The MUX{idx} divider output corresponds to the *DBUS*{idx} clock.

Example Setting *muxdiv0* to 5 will result in a clock of $\frac{eventclock}{5}$ in the MUX0 output, which is the DBUS0 clock.

4.1.5 Sequence RAM Setting

PV name	Description	Data Type
seqaddr	The Sequence RAM address for write operation	long (min=0, max=16383)
seqcode	Event code to be written in the Sequence RAM	long (min=0, max=255)
seqtime	Timestamp to be written in the Sequence RAM	ulong (32-bit)

The *seqaddr*, *seqcode*, and *seqtime* PVs are used for writing event-timestamp pairs to the *Sequence RAM*. The *seqaddr* PV specifies the address of the *Sequence RAM* to store the event-timestamp pair, while *seqcode* and *seqtime* specify the code and the timestamp to be written, respectively.

These PVs are used by the *Injection state machine* and MAY NOT BE SET MANUALLY during normal operation.

4.1.6 Sequence RAM Switch

PV name	Description	Data Type
seqcountRBV	The number of lines writen in current Sequence RAM	int (min=0, max=16383)
seqram_switch	Switch Sequence RAM (Any write operation causes switch)	long

The *seqcountRBV* PV corresponds to a counter which displays the number of write operations performed to the waiting *Sequence RAM*. Disabling the EVG or the *Event Sequencer*, as well as switching the *Sequence RAM*, clears *seqcountRBV*. The *seqcountRBV* PV is used by the *Injection state machine* to check if all the write commands were effective.

The *seqram_switch* PV switches the running Sequence RAM. The *Event Sequencer* starts in the *Stopped* state, i.e., no *Sequence RAM* is running. If the *Event Sequencer* is enabled, a switch will cause the first *Sequence RAM* (*SeqRAM 1*) to start running, and the *Event Sequencer* status to change to *SeqRAM 1*. A switch in this state will move the *Event Sequencer* to state *SeqRAM 2*, i.e., the *Event Sequencer* stops running the *SeqRAM 1* and starts to run the *SeqRAM 2*. Switching the *Sequence RAM* when the *SeqRAM 2* is running switches the *Event Sequencer* back to *SeqRAM 1* state. At any state, disabling the *Event Sequencer* moves it to the *Stopped* state. It is possible to write to the *SeqRAM 1* when the *Event Sequencer* status is *Stopped* or *SeqRAM 2*, while it is only possible to write to the *SeqRAM 2* when the *Event Sequencer* status is *SeqRAM 1*.

The *seqram_switch* PV is used by the *Injection state machine* and MAY NOT BE SET MANUALLY during normal operation.

4.1.7 Timestamp

PV name	Description	Data Type
TIMESRC	Timestamp PPS signal source	enum [0: "IDLE"][1: "DBUS"][2: "EXTERNAL"][3: "INTERNAL"]
TIMESRCRBV	"TIMESRC" PV Read Back Value	enum [0: "IDLE"][1: "DBUS"][2: "EXTERNAL"][3: "INTERNAL"]
UTC	Timestamp seconds counter	ulong (32-bit)
UTCRBV	"UTC" PV Read Back Value	ulong (32-bit)
subsecondRBV	Timestamp subsecond counter (event clock period resolution)	ulong (32-bit)

The *TIMESRC* PV defines the PPS signal source for the EVG module. When configured as *IDLE*, the UTC timestamp is not incremented and no *update UTC* event is sent by the EVG. When configured as *DBUS*, the EVG UTC is incremented by the *MUX6* divider output (*DBUS6*). When configured as *EXTERNAL*, the PPS signal comes from the corresponding external input in the EVG rear panel. When configured as *INTERNAL*, the PPS is obtained from the internal oscillator clock.

The *UTC* PV sets the module timestamp UTC, which is the number of seconds passed since some epoch.

The *subsecondRBV* PV holds the value of the timestamp subsecond counter, which is updated by the event clock.

> Writing *0* to the *UTC* field causes the EVG to broadcast its timestamp.
> Modifying the *TIMESRC* will cause the EVG to broadcast its timestamp.

4.1.8 Firmware version

PV name	Description	Data Type
FRMVERSION	Firmware version	char[12] (First 12 characters of the commit hash)

The *FRMVERSION* PV identifies the module's firmware version. It holds the first 12 characters of the commit hash.

4.1.9 Event Configuration

PV name	Description	Data Type
EVDESC{idx}	Description of event{idx}	string[40]
EVCODE{idx}	Event{idx} code	long (min=0, max=255)
EVTIME{idx}	Event{idx} timestamp (unit is event clock period)	long (min=100, max=2147483647)
EVDLYMODE{idx}	Delay mode for event{idx} (fix or incremental)	bool [0: "FIX"][1: "INCR"]
EVTRANSM{idx}	Transmission mode for event{idx}	bool [0: "ALWAYS"][1: "INJECTION"]

The *EVCODE{idx}* PV defines the code to be broadcast for the event being configured. Setting *EVCODE{idx}* to 0 (zero) makes the event an IDLE EVENT.

EVTIME defines the time instant (in event clock period units) after the *Sequence RAM* start for broadcasting the event.

EVDLYMODE configures the event to be fixed or incremental. Incremental events receive additional RF period steps depending on the target bucket.

EVTRANSM specifies whether the event is always transmitted, i.e., both when *SeqRAM 1* or *SeqRAM 2* is running, or just during the injection period, i.e, when *SeqRAM 2* is running. The PV *EVDESC* gives a description of the event's purpose.

4.1.10 Injection Control

PV name	Description	Data Type
STATEMACHINE	Injection state machine status	enum [0: "Initializing"][1: "Stopped"][2: "Running"][3: "Injecting"][4: "Restarting"][5: "Preparing to Run"][6: "Preparing to Inject"]
STOPSEQ	Stop Sequence RAM trigger	bool [0: "OFF"][1: "ON"]
STOPINJSEQ	Stop Injection trigger	bool [0: "OFF"][1: "ON"]
RUNSEQ	Run Sequence RAM trigger	bool [0: "OFF"][1: "ON"]
INJSEQ	Start injection trigger	bool [0: "OFF"][1: "ON"]
RESTSEQ	Restart Sequence RAM trigger	bool [0: "OFF"][1: "ON"]

The *STATEMACHINE* PV displays the *Injection state machine* current state.

The *RUNSEQ* trigger only works when the machine is in the *Stopped* state. When triggered, *RUNSEQ* causes the state machine to prepare for run *SeqRAM 1*. The state switches to *Preparing to Run* and the program writes the appropriate events to *SeqRAM 1*. If the write procedure succeeds, *SeqRAM 1* starts running and the state switches to *Running*.

The *INJSEQ* trigger only works when the machine is in the *Running* state. When triggered, it causes the state machine to prepare for injecting. The state switches to *Preparing to Inject* and the program writes to *SeqRAM 2*. If the write succeeds, *SeqRAM 2* starts to run and the state switches to *Injecting*. When the injection is finished, the state machine switches back to *Running* state.

During the injection process (*Injecting* state), the *STOPINJSEQ* trigger causes the injection to stop and the state machine to return to the *Running* state.

From any state, the *STOPSEQ* trigger causes the *Event Sequencer* to stop and the state machine to return to the *Stopped* state.

The *RESTSEQ* trigger is used whenever it is necessary to run the *SeqRAM 1* with new events, but the *SeqRAM 1* is running. The reset trigger stops the *Event Sequencer*, re-writes the events to the *SeqRAM 1*, and returns the state machine to the *Running* state.

4.1.11 Module support

PV name	Description	Data Type
download	Download all PV values to module parameters	bool [0: "OFF"][1: "ON"]
uploadT	Upload all module parameters to status and RBV PVs	bool [0: "OFF"][1: "ON"]
statdev	Device configuration status	bool [0: "WAITING"][1: "RUNNING"]
network	Network connection status	bool [0: "OFF"][1: "ON"]
SAVE	Trigger for the IOC save	bool [0: "OFF"][1: "ON"]

The *download* trigger causes all set point, select, or command PVs to be downloaded to the hardware. This feature is useful when configuring modules with saved parameters. On startup, the IOC will automatically download the saved parameters in the case that the module is disconfigured.

The *uploadT* trigger causes all status and read-back value PVs to upload their values from the hardware. On IOC startup, an upload will automatically happen for any module that is already configured.

The *statdev* PV is updated based on the module's *alive* status. Whenever the module is configured, *statdev* value is *RUNNING*. Otherwise, it is *WAITING*.

The *network* PV is set by the *EVG Setup state machine* and show the network connection status (ON/OFF).

The *SAVE* PV triggers the IOC save when set to 1.

4.2 EVR/EVE

4.2.1 Control and Status

PV name	Description	Data Type
enable	Enable/Disable EVR/EVE	bool [0: "OFF"][1: "ON"]
enableRBV	"enable" PV Read Back Value	bool [0: "OFF"][1: "ON"]
inhs	Interlock input status	bool [0: "DISSERTED"][1: "ASSERTED"]
link	Uplink status	[0: "UNLINK"][1: "LINK"]

The *enable* PV enables/disables the EVR/EVE. When the module is disabled, no output signal is generated.

The *inhs* PV displays the status of the interlock input. Whether the interlock input affects the EVR's SFP or EVE's FP outputs, depends on the interlock configuration of each output.

The *link* PV displays the uplink status. If the Event Generator is enabled and the uplink connection is working, the *link* status is *LINK*, otherwise it is *UNLINK*.

4.2.2 OTP{idx}_o {idx}=0...15

PV name	Description	Data Type
enable{idx}	Enable/Disable OTP{idx}_o	bool [0: "OFF"][1: "ON"]
enable{idx}RBV	"enable{idx}" PV Read Back Value	bool [0: "OFF"][1: "ON"]
pol{idx}	Polarity selection for OTP{idx}_o	bool [0: "NORMAL"][1: "INVERTED"]
pol{idx}RBV	"pol{idx}" PV Read Back Value	bool [0: "NORMAL"][1: "INVERTED"]
event{idx}	Event code mapping for OTP{idx}	long (min=0, max=255)
event{idx}RBV	"event{idx}" PV Read Back Value	long (min=0, max=255)
pulses{idx}	Number of pulses generated by OTP{idx}_o trigger	long (min=1, max=65535)
pulses{idx}RBV	"pulses{idx}" PV Read Back Value	long (min=1, max=65535)
width{idx}	OTP{idx}_o pulse width (event clock period resolution)	ulong (32-bit)
width{idx}RBV	"width{idx}" PV Read Back Value	ulong (32-bit)
delay{idx}	OTP{idx}_o pulse delay (event clock period resolution)	ulong (32-bit)
delay{idx}RBV	"{idx}" PV Read Back Value	ulong (32-bit)
time{idx}	Enable/Disable timestamps for OTP{idx} reception of configured event	bool [0: "NOT TIMESTAMPED"][1: "TIMESTAMPED"]
time{idx}RBV	"{idx}" PV Read Back Value	bool [0: "NOT TIMESTAMPED"][1: "TIMESTAMPED"]

The *enable*{*idx*} parameter enables/disables the OTP{idx} channel.

The *pol*{*idx*} parameter specifies the OTP{idx} polarity.

The *event*{*idx*} parameter specifies the event mapped to the OTP{idx} channel. When the mapped event is received by the module, it generates a trigger in OTP{idx} according to the OTP{idx} settings.

The pulses{idx} parameter allows for the generation of a pulse train, instead of a single trigger. The parameter specifies how many pulses must be generated upon event reception.

The *width*{*idx*} parameter defines the width of the trigger generated in *event clock period units*. In the case of a pulse train, *width*{*idx*} defines the width of each of the pulses in the pulse train. The pulse train duty cycle is always equal to $\frac{1}{2}$ (period of $2 \times$ *width*{*idx*}).

The *delay*{*idx*} parameter specifies the delay between the event reception and trigger generation. For pulse trains, the *delay*{*idx*} specifies the delay between the event reception and the pulse train start.

The *time*{*idx*} parameter enables/disables the timestamping functionality for the mapped event. If enabled, the event code and associated timestamp will be stored in the *Timestamp FIFO* when the mapped event is received.

4.2.3 OUT{idx} {idx}=0...7

PV name	Description	Data Type
itl{idx}	Enable/Disable OUT{idx} interlock function	bool [0: "OFF"][1: "ON"]
itl{idx}RBV	"itl{idx}" PV Read Back Value	bool [0: "OFF"][1: "ON"]
sel{idx}	OUT{idx} source selection	string ["OTP{x}", {x}=0...15]["Dbus{x}", {x}=0...7]
sel{idx}RBV	"sel{idx}" PV Read Back Value	
rfdly{idx}	OUT{idx} RF delay ((event clock period)/20 resolution)	long (min=0, max=20)
rfdly{idx}RBV	"rfdly{idx}" PV Read Back Value	long (min=0, max=20)
finedly{idx}	OUT{idx} RF delay (5ps resolution)	long (min=0, max=1023)
finedly{idx}RBV	"finedly{idx}" PV Read Back Value	long (min=0, max=1023)

The *itl*{*idx*} enables/disables the interlock function for OUT{idx}.

The *sel*{*idx*} selects the OUT{idx} signal source. It can be selected from any of the *OTP* and *Distributed Bus* channels.

The *rfdly*{*idx*} defines a delay with $\frac{event\ clock\ period}{20}$ resolution to be applied to the OUT{idx} trigger. *rfdly*{*idx*} can range from 0 to 20 steps. When set to 31, the OUT{idx} channel will set its RF delay based on special events sent by the EVG.

The *finedly*{*idx*} defines a fine delay with 5ps resolution to the OUT{idx} trigger.

4.2.4 RF Output (EVE)

PV name	Description	Data Type
cmsel	RF output configuration	enum [0: "OFF"][1: "RF/4"][2: "RF/2"][3: "RF"][4: "5*RF/4"][5: "10*RF/4"]
cmselRBV	"cmsel" PV Read Back Value	enum [0: "OFF"][1: "RF/4"][2: "RF/2"][3: "RF"][4: "5*RF/4"][5: "10*RF/4"]

The *cmsel* PV configures the EVE RF output, which can provide a clock multiple of the recovered *event clock*.

4.2.5 Timestamp

PV name	Description	Data Type
TIMESRC	Timestamp PPS signal source	enum [0: "IDLE"][1: "DBUS"][2: "EVENT"][3: "INTERNAL"]
TIMESRCRBV	"TIMESRC" PV Read Back Value	enum [0: "IDLE"][1: "DBUS"][2: "EVENT"][3: "INTERNAL"]
UTCRBV	Timestamp seconds counter	ulong (32-bit)
subsecondRBV	Timestamp subsecond counter (event clock period resolution)	ulong (32-bit)

The *UTCRBV* PV displays the module's UTC timestamp, which represents the number of seconds passed since some epoch. The *subsecondRBV* PV display the module's current subsecond timestamp, which is a counter incremented by the recovered *event clock* and reset by received PPS signals.

The *TIMESRC* parameter specifies the PPS signal souce for the EVR/EVE module. When it is set to *IDLE*, the module UTC does not change. When it is configured as *DBUS*, the module's UTC timestamp is incremented by the *DBUS6* clock received from the *UPLINK*. When it is configured as *EVENT*, the timestamp is incremented by the event 0x73 transmitted by the EVG. When it is configured as *INTERNAL*, the PPS is obtained from the internal oscillator frequency.

4.2.6 Timestamp Log

PV name	Description	Data Type
stoplog	Stop timestamping (stop hardware FIFO update)	bool [0: "OFF"][1: "ON"]
STOPLOGRBV	"stoplog" PV Read Back Value	bool [0: "OFF"][1: "ON"]
rstlog	Reset timestamp log hardware FIFO (restart timestamping after uplink disconnection)	bool [0: "OFF"][1: "ON"]
RSTLOGRBV	"rstlog" PV Read Back Value	bool [0: "OFF"][1: "ON"]
pull	Pull one timestamp from buffer	bool [0: "DO NOTHING"][1: "PULL"]
FULL	"Buffer is full" flag	bool [0: "NO"][1: "YES"]
EMPTY	"Buffer is empty" flag	bool [0: "NO"][1: "YES"]
LOGUTC	Timestamp UTC value at hardware FIFO output	ulong (32-bit)
LOGSUBSEC	Timestamp subsecond value at hardware FIFO output	ulong (32-bit)
LOGEVENT	Timestamp event value at hardware FIFO output	long (min=0, max=255)
UTCbuffer	Timestamp log UTC circular buffer	ulong (32-bit)
SUBSECbuffer	Timestamp log subsecond circular buffer	ulong (32-bit)
EVENTbuffer	Timestamp log event circular buffer	long (min=0, max=255)
LOGCOUNT	Number of elements in hardware FIFO	long (min=0, max=16384)
LOGSOFTCNT	Software buffers update counter (allow IOC client to track "new data" in buffers)	ulong (32-bit)

When *stoplog* is 1, the module's timestaping function remains stopped.

When *rstlog* is 1, it resets the module's *Timestamp FIFO*.

The *pull* PV causes a pair of event code and timestamp to be pulled from the *Timestamp FIFO* and stored in the IOC buffers. The IOC automatically pulls event-timestamp pairs from the FIFO whenever the FIFO is not empty.

The *FULL* and *EMPTY* flags indicate when the *Timestamp FIFO* is full or empty, respectively.

The *LOGUTC*, *LOGSUBSEC*, and *LOGEVENT* PVs receive the event-timestamp pair information when it is pulled from the hardware FIFO. Their purpose is to temporarily store the information for the

buffers, which cannot read it directly from the hardware.

The *UTCbuffer*, *SUBSECbuffer*, and *EVENTbuffer* PVs are circular buffers for storing the history of event and timestamps pulled from the hardware. The buffers read the information stored in the *LOGUTC*, *LOGSUBSEC*, and *LOGEVENT* PVs right after a pull is performed.

The *LOGCOUNT* displays how many items are currently stored in the hardware *Timestamp FIFO*.

> When the *UPLINK* signal is lost, the EVR/EVE stops the timestamping immediately and the *stoplog* flag remains active until *rstlog* is set.

4.2.7 Firmware Version

PV name	Description	Data Type
FRMVERSION	Firmware version	char[12] (First 12 characters of the commit hash)

The FRMVERSION PV identifies the module's firmware version. It holds the first 12 characters of the commit hash.

4.2.8 Configuration (EVR)

PV name	Description	Data Type
alive	Alive counter	ulong (32-bit)
funsel	STD-EVO configuration	enum [16: "FOUT"][17: "EVR"][18: "EVG"][32: "EVE"]
funselRBV	"funsel" PV Read Back Value	enum [16: "FOUT"][17: "EVR"][18: "EVG"][32: "EVE"]
clkmode	Setting according to uplink event clock frequency (16: 121MHz 135MHz)	enum [11: "60-62.5 MHz"][12: "63-77 MHz"][13: "77.5-91.5 MHz"][14: "92-106 MHz"][15: "106-120.5 MHz"][16: "121-135 MHz"]
clkmodeRBV	"clkmode" PV Read Back Value	enum [11: "60-62.5 MHz"][12: "63-77 MHz"][13: "77.5-91.5 MHz"][14: "92-106 MHz"][15: "106-120.5 MHz"][16: "121-135 MHz"]

The *Configuration* parameters define the STD-EVO module basic functionality. The *funsel* parameter is used to configure the module as EVG, EVR, or FOUT.

The *alive* counter works as a heartbeat, and starts to count as soon as the module receives one of the possible configuration options: EVG, EVR, or FOUT. The *EVRE Setup state machine* uses the *alive* information to determine the status of the network connection.

On startup, the *EVRE Setup state machine* will set the *funsel* parameter in the case of the module being disconfigured.

The *clkmode* parameter must be configured according to the *UPLINK* signal frequency.

4.2.9 Configuration (EVE)

PV name	Description	Data Type
alive	Alive counter	ulong (32-bit)
funselRBV	Module configuration status	enum [16: "FOUT"][17: "EVR"][18: "EVG"][32: "EVE"]
clkmode	Setting according to uplink event clock frequency (16: 121MHz 135MHz)	enum [11: "60-62.5 MHz"][12: "63-77 MHz"][13: "77.5-91.5 MHz"][14: "92-106 MHz"][15: "106-120.5 MHz"][16: "121-135 MHz"]
clkmodeRBV	"clkmode" PV Read Back Value	enum [11: "60-62.5 MHz"][16: "63-77 MHz"][17: "77.5-91.5 MHz"][18: "92-106 MHz"][19: "106-120.5 MHz"][16: "121-135 MHz"]

The *Configuration* parameters define the STD-EVO module basic functionality. The *funsel* parameter is used to configure the module as EVG, EVR, or FOUT.

The *alive* counter works as a heartbeat, and starts to count as soon as the module receives one of the possible configuration options: EVG, EVR, or FOUT. The *EVRE Setup state machine* uses the *alive* information to determine the status of the network connection.

The *clkmode* parameter must be configured according to the *UPLINK* signal frequency.

4.2.10 Module support

PV name	Description	Data Type
download	Download all PV values to module parameters	bool [0: "OFF"][1: "ON"]
uploadT	Upload all module parameters to status and RBV PVs	bool [0: "OFF"][1: "ON"]
statdev	Device configuration status	bool [0: "WAITING"][1: "RUNNING"]
network	Network connection status	bool [0: "OFF"][1: "ON"]
SAVE	Trigger for the IOC save	bool [0: "OFF"][1: "ON"]

The *download* trigger causes all set point, select, or command PVs to be downloaded to the hardware. This feature is useful when configuring modules with saved parameters. On startup, the IOC will automatically download the saved parameters in the case that the module is disconfigured.

The *uploadT* trigger causes all status and read-back value PVs to upload their values from the hardware. On IOC startup, an upload will automatically happen for any module that is already configured.

The *statdev* PV is updated based on the module's *alive* status. Whenever the module is configured, *statdev* value is *RUNNING*. Otherwise, it is *WAITING*.

The *network* PV is set by the *EVG Setup state machine* and show the network connection status (ON/OFF).

The *SAVE* PV triggers the IOC save when set to 1.

5 Startup Script

Instructions for configuring the startup script are provided in this section. The startup script is responsible for instantiating and initializing the IOC components. The script path is *<ioc directory>/iocBoot/ioctiming/st.cmd*.

5.1 AsynDriver IP port configuration

The IOC communication with timing modules (device support) is accomplished by the StreamDevice tool, which uses asynDriver for low-level support, i.e., implementation of UDP communication.

TCP/IP or UDP/IP connections are configured with the *drvAsynIPPortConfigure* command in the IOC startup script:

```
drvAsynIPPortConfigure("portName","hostInfo",priority,noAutoConnect,noProcessEos)
```

where:

portName The desired port name. The port name is used later in the startup script when loading record databases. For example, a port used for connecting to the EVG module could be called *evgPort*, and later the EVG record database would receive the *evgPort* as parameter.

hostInfo The Internet host name, port number, optional local port number, and optional IP protocol of the device. The format is "<host>:<port>[:localPort] [protocol]". For the Timing System IOC the protocol used is UDP and the local port is required. For an example UDP port, hostInfo would look like "192.168.1.120:60111:60111 udp".

priority Priority at which the asyn I/O thread will run. If this is zero or missing, then epicsThreadPriorityMedium is used.

noAutoConnect Zero or missing indicates that portThread should automatically connect. StreamDevice automatically connects independently of this selection.

noProccessEos If 0 then asynInterposeEosConfig is called specifying both processEosIn and processEosOut.

The value adopted for the priority, noAutoConnect, and noProccessEos parameters is 0. Therefore, the port configuration may follow the following syntax:

```
drvAsynIPPortConfigure("portName","hostInfo",0,0,0)
```

5.2 Loading Records

Record databases need to be loaded before *iocInit*.

The record databases are loaded using the following syntax:

```
dbLoadRecords("<database file path>", "<macro substitutions>")
```

A given database file can be loaded many times with different arguments (macro substitution strings) in order to load different instances of the records defined in the file.

```
dbLoadRecords("$TOP/db/evr.db", "Dev=EVR, Idx=1, PORT=EVR1-port")
dbLoadRecords("$TOP/db/evr.db", "Dev=EVR, Idx=2, PORT=EVR2-port")
dbLoadRecords("$TOP/db/evr.db", "Dev=EVR, Idx=3, PORT=EVR3-port")
```

In the example above, three copies of each record defined in "evr.db" are loaded, but with different macro substitutions. For example, the records loaded by the first command could have prefix "EVR-1:", while the records resulting from the second and third commands would have prefixes "EVR-2" and "EVR-3" respectively. The port passed for each command was also different, connecting each group of records to a different device.

5.3 Loading Sequencer Programs

Sequencer programs need to be called after *iocInit*.

Sequencer programs are loaded using the follwing syntax:

```
seq <program name>,"<macro substitutions>"
```

A given sequencer program can be loaded many times with different arguments (macro substitution strings) in order to load different instances of the state machines defined in the file.

```
seq sncEVRESetup, "Sec=AS, Sub=Inj, Dis=TI, Dev=EVR, Idx=1"
seq sncEVRESetup, "Sec=AS, Sub=Inj, Dis=TI, Dev=EVR, Idx=2"
seq sncEVRESetup, "Sec=AS, Sub=Inj, Dis=TI, Dev=EVR, Idx=3"
```

5.4 EVG related Databases and Sequencer Programs

The steps for loading the records and sequencer programs involved in the tasks performed by the EVG module are described below.

5.4.1 EVG database instantiation

The records associated with hardware read and write operations are instantiated in the EVG.db file. In order to load the file, the following syntax is used:

```
dbLoadRecords("$TOP/db/evg.db", "Sec=sec-example, Sub=sub-example, Dis=dis-example, Dev=EVG, Idx=1, ITBL_LEN=2000, PORT=EVG1-port")
```

Sec, Sub, Dis, Dev, and Idx are the prefix fields for the given device according to the Sirius Naming System.

The *ITBL_LEN* argument defines the length of the array used for storing the bucket list.

The *PORT* argument defines the port associated with the EVG hardware module. The port name used here is the same defined for the EVG module when configuring the AsynDriver IP port.

5.4.2 Event instantiation

The events generated by the EVG are instantiated by loading the *Events.db* file. The example below illustrates the command syntax for loading a event:

```
dbLoadRecords "$TOP/db/Events.db", "Sec=sec-example, Sub=sub-example, Dis=dis-example, Dev=dev-example, Idx=1, num=0, desc='example event', code=0x01, time=100, mode=0, transm=0"
```

Sec, Sub, Dis, Dev, and Idx are the prefix fields for the given device according to the Sirius Naming System. The prefixes used for the events MUST BE THE SAME USED FOR THE EVG MODULE.

The *num* argument corresponds to the index of the event that will appear in the corresponding PVs' names. The *num* argument must be unique, positive and smaller than the number of events defined. For instance, if there are 4 events defined, they must receive indexes *0*,*1*,*2*, and *3*. This is necessary because of the way the injection state machine obtains the event parameters.

The *desc* argument receives a description of the event purpose (maximum of 40 characters). The *code* argument receives the code associated to the event. The *time* argument receives the timestamp for the given event. The *mode* argument specifies if the given event timestamp must be incremented according to the bucket list. The *transm* argument specifies if the event must be in both Sequence RAMs, or only in the injection *Sequence RAM* (*SeqRAM 2*).

After the event records have been loaded, the number of events loaded MUST BE PASSED to the Injection State Machine. The *ev_num* argument of the Injection State Machine call must be set equal to the number of events instantiated.

Example *seq sncSeqRAM, "Sec=example, Sub=, Dis=, Dev=, Idx=, ev_num=4"*

5.4.3 EVG Setup Sequencer Program

The EVG Setup sequencer program is responsible for downloading the IOC currently saved parameters for the EVG module to the hardware in the case that the module is disconfigured when the IOC is initialized. If the module is already configured when the IOC is initialized, the Setup state machine uploads the hardware parameters to the IOC.

The following command syntax is used for initializing the EVG Setup sequencer program:

```
seq sncEVGSetup, "Sec=AS, Sub=Inj, Dis=TI, Dev=EVG, Idx=1"
```

Sec, Sub, Dis, Dev, and Idx are the prefix fields for the given device according to the Sirius Naming System.

5.4.4 Injection State Machine

The Injection sequencer program is responsible for controlling the event distribution associated with the injection process. The program does so by appropriately controlling the *Event Sequencer* and writing to the *Sequence RAM*s.

The following command syntax is used for initializing the Injection sequencer program:

```
seq sncSeqRAM, "Sec=AS, Sub=Inj, Dis=TI, Dev=EVG, Idx=1, ev_num=4"
```

Sec, Sub, Dis, Dev, and Idx are the prefix fields for the given device according to the Sirius Naming System.

The *ev_num* argument is used for passing the number of existing events to the Injection sequencer program. This number along with the name prefix fields is used by the program to figure out the events PVs names. Passing the wrong value will cause the program to have problems during execution.

5.5 EVR/EVE related Databases and Sequencer Programs

The steps for loading the records and sequencer programs involved in the tasks performed by the EVR or EVE modules are described below.

5.5.1 EVR/EVE Database instantiation

The records associated with hardware read and write operations are instantiated in the EVR.db and EVE.db files for the EVR and EVE modules, respectively. In order to load the files, the following command syntax is used:

```
dbLoadRecords("$TOP/db/evr.db", "Sec=AS, Sub=Inj, Dis=TI, Dev=EVR, Idx=1,
PORT=EVR1-port")
```

Sec, Sub, Dis, Dev, and Idx are the prefix fields for the given device according to the Sirius Naming System.

The *PORT* argument defines the port associated with the EVR/EVE hardware module. The port name used here is the same defined for the given EVR/EVE module when configuring the AsynDriver IP port.

5.5.2 EVRE Setup Sequencer Program

The EVRE Setup sequencer program is responsible for downloading the IOC currently saved parameters for a EVR or EVE module to the hardware in the case that the module is disconfigured when the IOC is initialized. If the module is already configured when the IOC is initialized, the Setup state machine uploads the hardware parameters to the IOC.

The following command syntax is used for initializing the EVRE Setup sequencer program:

```
(EVR example) seq sncEVRESetup, "Sec=AS, Sub=Inj, Dis=TI, Dev=EVR, Idx=1"
(EVE example) seq sncEVRESetup, "Sec=AS, Sub=Inj, Dis=TI, Dev=EVE, Idx=1"
```

Sec, Sub, Dis, Dev, and Idx are the prefix fields for the given device according to the Sirius Naming System.

5.6 Specifying the Access Security File

The Access Security File specifies the read/write rights for IOC clients. The following command is used for specifying the file location:

```
asSetFilename("<file path>")
```

Example *asSetFilename("$(TOP)/accessSecurityFile.acf")*

5.7 Autosave settings

The autosave tool allow the PV values to be restored when the IOC is restarted.

5.7.1 Autosave files path specification

The path for storing the save files is defined using the following command syntax:

```
set_savefile_path("<save path>", "<sub path>")
```

Example *set_savefile_path("$(TOP)", "autosave/save_files")*

The request files specify the PVs to be saved. The path for storing the request files is defined using the following command syntax:

```
set_requestfile_path("<save path>", "<sub path>")
```

Example *set_requestfile_path("$(TOP)", "autosave/request_files")*

The command for defining autosave files paths must be issued before *iocInit*.

5.7.2 Restoring PV values

The commands presented below can be used to restore the PV values stored in a given save file.

For restoring the PV values WITHOUT PROCESSING the associated records, the following command is used:

```
set_pass0_restoreFile("<save file name>", "<macro substitution strings>")
```

Example *set_pass0_restoreFile("EVG_save.sav", "Sec=AS, Sub=Inj, Dis=TI, Dev=EVG, Idx=1")*

For restoring the PV values AND PROCESS the associated records, the following command is used:

```
set_pass1_restoreFile("<save file name>", "<macro substitution strings>")
```

Example *set_pass1_restoreFile("Event0_save.sav", "Sec=AS, Sub=Inj, Dis=TI, Dev=EVG, Idx=1")*

The command for restoring PVs must be issued before *iocInit*.

5.7.3 Triggering the save action

The autosave tool can be configured to save the IOC PVs whenever the value of a given PV changes. The following command is used to create a set of PVs that are saved once a determined PV changes:

```
create_triggered_set("<request file path>", "<trigger PV name>", "<macro substitutions>")
```

Example *create_triggered_set("EVG.req", "AS-Inj:TI-EVG1:SAVE", "Sec=AS, Sub=Inj, Dis=TI, Dev=EVG, Idx=1")*

The command for creating a triggered save set must be issued after *iocInit*.

6 Common Errors and Causes

6.1 EVG RF status indicates loss, although the RF signal is fine

- If the *rfdiv* PV is set to a value that result in the event clock being out of range, the RF status is going to indicate loss.
- If the module loses the RF signal, even for a short period, and recovers it afterwards, the *rfdiv* value must be set again in order to allow the RF status to return to *normal*.

6.2 Injection State Machine fails to run

Error message: *"Error: Failed to switch or to write to SeqRAM 1."*

- If the module RF status state is *Loss or out of range*, the Injection State Machine is not capable of switching to the SeqRAM 1. After the write operation timeout, the state machine prints the error message and returns to the *Stopped* state.

6.3 Injection State Machine gets stuck in the *Preparing to Run* state

- In the st.cmd file, providing a number of events to the Injection State Machine that is greater than the number of events created causes the EVG Injection State Machine to try to read the inexistent PVs indefinitely and to never be able to run the Sequence RAM.

6.4 Injection State Machine fails to inject

Error message: *"Error: Failed to switch or to write to SeqRAM 2."*

- The injection fails if the AC Line signal is missing or disabled.
- Too few events in the Sequence RAM 2 can cause the EVG Injection State Machine to print a wrong injection failure message. If the Sequence RAM 2 transmits only 1 event (or none), it switches to SeqRAM 2 and back to SeqRAM 1 before the IOC state machine can detect the injection. Since from the state machine point of view the EVG never switched from 'Running' to 'Injecting', i.e., it kept running the Sequence RAM 1 as if ignoring the injection command, an error message is printed, although, in actuality, the EVG has successfuly injected (run SeqRAM 2).

6.5 STOPLOG cannot be turned off

- When the *Event Receiver* loses the uplink signal, the *stoplog* parameter is automatically set to 1, in order to freeze the Timestamp FIFO. When this happens, it is necessary to set *rstlog* to 1 (and back to 0 afterwards), in order to reset the Timestamp FIFO and allow *stoplog* to be set to 0.

6.6 Event settings ignored

- If the number of events provided to the Injection State Machine is smaller than the actual number of events created, only the events with smaller indexes will be recognized by the state machine.

6.7 Communication timout

Error message: *"<date><time><port name><record name>: No reply from device within <timeout>ms"*

- StreamDevice generates an error message when a reply timout happens. Common communication problems causes are incorrect device address specification in the st.cmd file, and device physical disconnection.

6.8 IOC State Machines present strange behavior

- If two Timing IOCs are runing at the same time, their state machines actions will conflict and cause unexpected behavior.

6.9 IOC crash at initialization

Using *StreamDevice 2-7-7* fixes this problem

- When initializing the autosave tool in the st.cmd file, and setting the file restore configuration, if the command *set_pass0_restoreFile* or *set_pass1_restoreFile* receives too large macro substitution strings, the IOC can crash. Since the save files can be restored without passing any macro substitution string to the function, the problem can be avoided by do not using substitution strings in these commands.

- When configuring the '.db' files of the IOC, providing too large macro substitution strings to a streamDevice protocol can cause the IOC to crash. The way StreamDevice is implemented restricts the maximum length of a record name passed to the protocol when using redirection to records together with protocol arguments. If the string passed is too large the IOC terminates with segmentation fault.