CNPEM
Centro Nacional de Pesquisa
em Energia e Materiais


Laboratório Nacional
de Luz Síncrotron

# STD-EVO Hardware Manual
# 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 manual is intended for people who need information about the STD-EVO hardware. Information about the Timing System structure and operation, firmware, or software can be found in the corresponding manuals.

# Contents


**1 Important Parameters** **2**

**2 Hardware Specification** **2**

**3 STD-EVO Hardware** **4**
- 3.1 STD-EVO/EVG . . . 4
  - 3.1.1 Event Sequencer . . . 4
  - 3.1.2 Distributed Bus . . . 6
  - 3.1.3 Timestamp . . . 6
  - 3.1.4 Registers . . . 7
- 3.2 STD-EVO/EVR . . . 12
  - 3.2.1 Clock Recovery . . . 13
  - 3.2.2 Trigger Generation . . . 13
  - 3.2.3 SFP Ouputs . . . 14
  - 3.2.4 POF Ouputs . . . 14
  - 3.2.5 Timestamp . . . 14
  - 3.2.6 Registers . . . 16
- 3.3 STD-EVO/FOUT . . . 22
  - 3.3.1 Registers . . . 23

**4 Ethernet Network Interface** **24**
- 4.1 Network configuration . . . 24
- 4.2 Register Read/Write . . . 25

**A Special Events** **26**

# 1 Important Parameters

- 9 SFP connectors in the front panel (OUT0 - OUT7 and UPLINK).
- External RF clock can be divided by 1 - 16 in STD-EVO.
- External RF clock frequency range is 60 - 540MHz.
- Event clock frequency range is 60 - 135MHz.

# 2 Hardware Specification

STD-EVO is a 19 inches 1U module.
110/220V 50/60Hz AC power supply.
**NOTE: When configured as EVG, RF input should be always available before power on.**

Figure 1: STD-EVO

Table 1: STD-EVO front panel connectors

| **Connector** | **Type** | **Description / Specification** |
|---|---|---|
| RF IN | LEMO EPL.00.250.NTN | RF input<br>0 - 10dBm, 60MHz - 540MHz<br>50ohm impedance |
| AC IN / INH | LEMO EPL.00.250.NTN | AC line input (as EVG)<br>Interlock input (as EVR)<br>TTL level<br>50ohm impedance |
| UPLINK | LC Duplex | Fiber for uplink |
| OUT0 - OUT7 | LC Duplex | Fibers for downlink |

Table 2: STD-EVO front panel LEDs

| **LED** | **Type** | **Description / Specification** |
|---|---|---|
| EVG | Green LED | STD-EVO configured as EVG |
| EVR | Green LED | STD-EVO configured as EVR |
| FOUT | Green LED | STD-EVO configured as FOUT |
| FPGA | Green LED | FPGA downloaded |
| INH | Red LED | Interlock input activated (as EVR)<br>RF input error (as EVG) |
| ENA | Green LED | STD-EVO enabled |
| EVT | Yellow LED | (Blink) Event code transmitted (as EVG)<br>(Blink) Event code received (as EVR or FOUT) |
| LINK | Green LED | Uplink established |

Table 3: STD-EVO rear panel connectors

| **Connector** | **Type** | **Description / Specification** |
|---|---|---|
| ETHERNET | RJ45 | 10/100Mbit Ethernet port |
| RST | Button | Reset Ethernet |
| OTP0 | HFBR-4531/4532 | OTP0 output (Agilent HFBR-1528) |
| OTP1 | HFBR-4531/4532 | OTP1 output (Agilent HFBR-1528) |
| OTP2 | HFBR-4531/4532 | OTP2 output (Agilent HFBR-1528) |
| OTP3 | HFBR-4531/4532 | OTP3 output (Agilent HFBR-1528) |
| OTP4 | HFBR-4531/4532 | OTP4 output (Agilent HFBR-1528) |
| OTP5 | HFBR-4531/4532 | OTP5 output (Agilent HFBR-1528) |
| OTP6 | HFBR-4531/4532 | OTP6 output (Agilent HFBR-1528) |
| OTP7 | HFBR-4531/4532 | OTP7 output (Agilent HFBR-1528) |
| OTP8 | HFBR-4531/4532 | OTP8 output (Agilent HFBR-1528) |
| OTP9 | HFBR-4531/4532 | OTP9 output (Agilent HFBR-1528) |
| OTP10 | HFBR-4531/4532 | OTP10 output (Agilent HFBR-1528) |
| OTP11 | HFBR-4531/4532 | OTP11 output (Agilent HFBR-1528) |

Table 4: STD-EVO rear panel LEDs

| **LED** | **Type** | **Description / Specification** |
|---|---|---|
| PWR | Green LED | Power on |

# 3 STD-EVO Hardware

The STD-EVO module can be configured to perform one of three different roles in the timing system. It can be configured as *Event Generator* (EVG), *Event Receiver* (EVR), or *Fanout* (FOUT). The following sections describe each of the configurations in detail.


3.1 STD-EVO/EVG . . . . . 4
3.1.1 Event Sequencer . . . . . 4
3.1.2 Distributed Bus . . . . . 6
3.1.3 Timestamp . . . . . 6
3.1.4 Registers . . . . . 7
3.2 STD-EVO/EVR . . . . . 12
3.2.1 Clock Recovery . . . . . 13
3.2.2 Trigger Generation . . . . . 13
3.2.3 SFP Ouputs . . . . . 14
3.2.4 POF Ouputs . . . . . 14
3.2.5 Timestamp . . . . . 14
3.2.6 Registers . . . . . 16
3.3 STD-EVO/FOUT . . . . . 22
3.3.1 Registers . . . . . 23


## 3.1 STD-EVO/EVG

When configured as EVG, the STD-EVO is an *Event Generator*. In this configuraton, it acts as the Timing System master, being able to provide events and clocks to the Timing System. The STD-EVO module has three inputs: *RF IN*, *ACIN INH*, and *UPLINK*. The signal associated with each input for the EVG configuration is described below.

The *RF IN* input receives the RF signal, allowing the *Event Generator* to be synchronized with the RF frequency. The *RF IN* signal also defines the *event clock* frequency, which is the RF frequency divided by the *RF divider*, which is user-defined. The *event clock* defines the parallel clock for distributing the Timing System data frames, i.e., the frequency with which data frames are broadcast to the Timing System. Each data frame is composed of two bytes (see figure 2). One byte contains an event code, which is used by the *Event Receivers* (EVR/EVE) for generating triggers. The other byte contains clock information, and will be referred to as *Distributed Bus* (*DBUS*). Each bit in the *Distributed Bus* contains clock information about one of the EVG's channels. Therefore, the EVG (and consequently the Timing System) is able to transmit 8 channels' clocks and a trigger simultaneously.

| *Event code* | *DBus* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 8 bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Figure 2: Timing System Data Frame

The *ACIN* input (ACIN INH) receives the mains signal when the module operates as EVG. In this case, the *ACIN* signal will be referred to as *AC Line*. The *AC Line* is used for sychronizing the event distribution with the mains signal. This synchronization is accomplished by the *Synchronizer* submodule, which is described in section 3.1.1.

The *UPLINK* input is able to receive a data frame whose event code is immediately broadcast to the Timing System.

The STD-EVO module has 8 SFP outputs in the front panel which, in EVG mode, transmit the data frames containing the event code and the *Distributed Bus*. The *Event Sequencer* is the EVG submodule responsible for storing event codes and distributing them in the associated timestamps. Details of the *Event Sequencer* and other EVG submodules and functions will be provided later in this section.

Another feature of the EVG is *Timestamp Distribution*. The EVG is able to broadcast its timestamp to the Timing System and also to transmit a PPS signal. The UTC timestamp and PPS source are user-configurable.

### 3.1.1 Event Sequencer

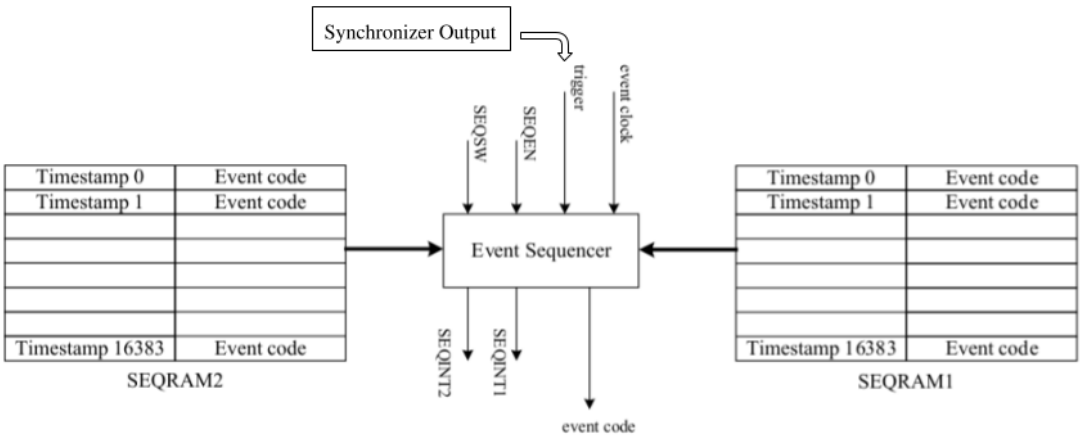
The *Event Sequencer* is able to transmit event codes in the Timing System in previously determined instants of time with resolution of *event clock* period. Additional and finer delays can be defined in the *Event Receivers*. The event codes and corresponding timestamps are stored in a *Sequence RAM* (*SeqRAM*), which can store up to 16384 sets of event code and timestamp (see figure 3). In order to write a set of event code and timestamp to a given address of the *SeqRAM*, the SeqRAM Setting Register is used, in which *SEQADDR* is the *Sequence RAM* address, *SEQCODE* is the event code to be written, and *SEQTIME* is the associated timestamp. The *SeqRAM* reading and distribution of stored event codes is started by a trigger, which can be selected from *Synchronizer* output, VME writing access or the output of the *Event FIFO*. The trigger is also responsible for restarting the *Event Sequencer* timestamp, which is a counter incremented by the *Event Sequencer* running clock, which is the *event clock*. When the *Event Sequencer* clock count is equal to the timestamp stored in the current position of the *SeqRAM*, the corresponding event code is transmitted, and then the timestamp and event code move to the next contents in the *SeqRAM*. The *Event Sequencer* timestamp has length of 32 bits. When the current timestamp reaches 0xFFFFFFFF, the *Event Sequencer* stops and waits for the next trigger.

Some event codes are used for commanding the *Event Sequencer*. The event code 0x70 makes the *Event Sequencer* stop at its current position in the *SeqRAM* and wait for the next trigger to continue. The code 0x7F makes the *Event Sequencer* move to the beginning of the *SeqRAM* and wait for a trigger.

There are two *SeqRAMs* in the *Event Sequencer*, which can be alternately used. The switching between the two *SeqRAMs* happen in two cases. When a write operation is executed to the *SeqRAM Switch Register* (*SEQSW* signal), or when the running *SeqRAM* reads the event code 0x7E. In both cases, the switch will cause the *Event Sequencer* to start reading the other *SeqRAM* from the beggining.

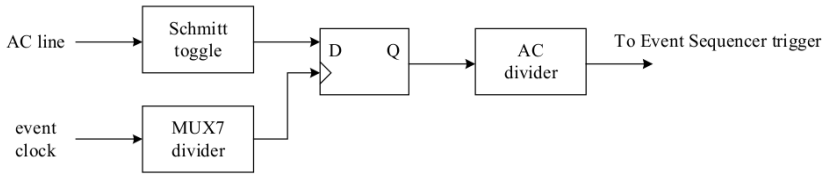
The complete list of special event codes is available in the Appendix A.

Figure 3: Event Sequencer



**Synchronizer** The synchronizer is responsible for providing the *Event Sequencer* trigger. This trigger is the output of a D flip flop whose inputs are the *AC Line* frequency and the *MUX 7* divider output (see figure 4). The *MUX 7* divider allows for the synchronization of the *AC Line* with any submultiple of the *event clock*. Setting the *MUX 7* divider to the *Coincidence Number*, synchronizes the *Event Sequencer* trigger with both the *AC Line* and the Coincidence Frequency. The *AC divider* following the D flip flop output allows the trigger frequency to be reduced to any submultiple of the mains frequency (the slowest of the synchronization frequencies).

Figure 4: Synchronizer




### 3.1.2 Distributed Bus

The *Distributed Bus* contains 8 clock channels, each represented by a bit in the data frame. The EVG has 8 dividers, *MUX0 - MUX7*, whose outputs are mapped to the *Distributed Bus* channels 0 - 7 respectively. Since the dividers input is the *event clock*, each *MUX* divider output, i.e., *Distributed Bus* channel, is able to transmit a different submultiple of the *event clock*. The configuration of the *MUX* dividers and respective channels is done through the MUXx Register.

### 3.1.3 Timestamp

The EVG is capable of distributing timestamps to the Timing System, allowing *Event Receivers* to have accelerator-wide synchronized timestamping. The parameters and timestamp information are written/read in the *Timestamp Register*.

The Timing System 64-bit timestamp is divided in two parts. The first 32 bits hold the UTC data, containing the number of seconds passed since some epoch. This information is set by software (*UTC* field) and is incremented by a user-configured PPS (Pulse Per Second) source. The EVG module is able to receive an external PPS signal (e.g., from a GPS), obtain it from the *MUX 6* RF frequency divisor output, or directly from its internal oscillator. The PPS source is configured by the *TIMESRC* field. There are a few circuntances when the EVG UTC timestamp is broadcast to the Timing System. The first is when a broadcast command is sent to the EVG, i.e., the value 0 is written to its *UTC* field. The UTC timestamp will NOT be modified by the written 0, but the current timestamp will be broadcast. The second option for triggering a broadcast is to write a non-zero new value to the EVG's *UTC* field. This will cause the timestamp to change and trigger the broadcast. The last case of timestamp broadcast happens when the EVG PPS signal source is changed. For broadcasting the timestamp, the EVG first sends the 0x74 event code, which tells the system that the UTC timestamp is going to be transmitted. The following 4 data frames contain UTC information in the event code byte. The 0x74 event resets the receivers' timestamps and guarantees that the UTC information will not be interpreted as actual event codes.

Every second, the EVG sends an *increment UTC* event (0x73), which may or may not be used by the *Event Receivers*. If the Timing System network is uniform, the update signals reach all receivers at the same time.

The second part of the Timing System timestamp is a 32-bit subsecond counter (*SUBSECOND field*). The subsecond increment signal is the *event clock*, and thus, it has resolution of event clock period. Every UTC update or increment, the *SUBSECOND* counter resets.

### 3.1.4 Registers

| Address | Register Name | Description |
|---|---|---|
| **0** | Control and Status Register | General settings and status. Enable/disable module, *Event Sequencer* status, *Event Sequencer* enable/disable, and RF signal status. |
| **40** | AC line Register | Mains signal input configuration. *AC Line* enable/disable, and *AC divider* setting. |
| **41 - 48** | MUX0 - MUX7 Register | *MUX* channels (*Distributed Bus* signals) settings. MUX enable/disable, and MUX divider setting. |
| **49** | SeqRAM Setting Register | Parameters for writing to the *Sequence RAM*. *SeqRAM* address, event code, and timestamp. |
| **50** | SeqRAM Switch Register | Switch *Sequence RAM*. *Sequence RAM* write operations counter. |
| **51** | Timestamp Register | Timestamp settings. Define PPS signal source and UTC timestamp. SUBSECOND reading. |
| **62** | Firmware Version Register | 12-byte code for current firmware version. |
| **63** | Configuration Register | Main STD-EVO configuration options. Function selection (EVG, EVR, FOUT), and *RF divider* setting. |

**Control and Status Register** [0]

RegC

| Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 |
|---|---|---|---|---|---|---|---|
| | | | RFINS | SEQS | | | |

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| | | | | | | SEQEN | EVGEN |

EVGEN — Enable/Disable EVG. Disabling the EVG puts the *Event Sequencer* in Stopped (idle) state, and disables *MUX* channels.

SEQEN — Enable/Disable the *Event Sequencer*. Disabling the *Event Sequencer* will return it to Stopped (idle) state.

SEQS — *Event Sequencer* status.

- 0 Stopped
- 1 SEQRAM1 running
- 2 SEQRAM2 running

RFINS — RF input status.

- 0 Loss or out of range
- 1 Normal

### AC line Register [40]

RegA

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| | | | | | | | EN |

RegC

| Bit 31 | Bit 0 |
|---|---|
| DIV | |

EN — Enable/Disable AC Line. AC Line and event clock are the sources for the *Event Sequencer* trigger. While the AC Line is disabled, the *Event Sequencer* will not be triggered.

DIV — AC Line divider. The divider to be applied for AC Line signal. Since *ACIN* (input for AC Line) receives the mains signal, the AC Line frequency is equal to mains's (50/60Hz) divided by the AC divider.

Prescaler = DIV + 1.

Example: For DIV = 0, AC Line input is divided by 1.

### MUXx Register [41-48]

RegA

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| | | | | | | | EN |

RegC

| Bit 31 | Bit 0 |
|---|---|
| DIV | |

EN — Enable/Disable MUXx channel. Disabling the *MUXx* channel causes the corresponding *Distributed Bus* bit to remain in zero.

DIV — MUXx divider. The divider to be applied for MUXx channel. The MUXx channel clock is equal to the *event clock* divided by the MUXx divider.

Prescaler = DIV + 1.

Example: For DIV = 0, MUXx input is divided by 1.

### SeqRAM Setting Register [49]

RegA

| Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 |
|---|---|---|---|---|---|---|---|
| | | SEQADDR | | | | | |

| Bit 7 | | | | | | | Bit 0 |
|---|---|---|---|---|---|---|---|
| SEQADDR | | | | | | | |

RegB

| Bit 7 | | | | | | | Bit 0 |
|---|---|---|---|---|---|---|---|
| SEQCODE | | | | | | | |

RegC

| Bit 31 | | | | | | | Bit 0 |
|---|---|---|---|---|---|---|---|
| SEQTIME | | | | | | | |

SEQADDR — The address of the waiting *Sequence RAM* to receive the event code and timestamp values.

SEQCODE — The event code to be written to the *Sequence RAM* specified address.

SEQTIME — The timestamp to be written to the *Sequence RAM* specified address.

### SeqRAM Switch Register [50]

RegC

| Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 |
|---|---|---|---|---|---|---|---|
| | | SEQCOUNT | | | | | |

| Bit 7 | | | | | | | Bit 0 |
|---|---|---|---|---|---|---|---|
| SEQCOUNT | | | | | | | |

A write to this register switches the *Sequence RAM*.

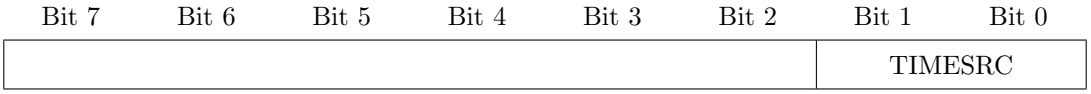
SEQCOUNT — The number of elements in the waiting *Sequence RAM*. This counter is incremented by write operations to the *SeqRAM*, and reset by disabling the *Event Sequencer* or the EVG. Read only.

**Timestamp Register [51]**

RegA


RegB


RegC




UTC — The timestamp UTC field, which is a 32-bit counter to store the number of seconds passed since some epoch. The counter is incremented by the source defined in TIMESRC. A write to UTC will change the current UTC value and cause the EVG to broadcast the new timestamp. Writing zero to UTC causes the EVG to broadcast the UTC timestamp without changing its current value. The UTC broadcast, in addition to updating *Event Receivers*'s UTCs, is reponsible for resetting all *Event Receivers*'s SUBSECOND.

SUBSECOND — The timestamp subsecond field, which is a 32-bit counter to store the subsecond portion of the Timing System timestamp. SUBSECOND is incremented by the *event clock*, and thus, it has *event clock* period resolution. The SUBSECOND is reset by any change of the UTC value, i.e., register access or PPS signal. The timestamp subsecond portion is not broadcast to the Timing System.

TIMESRC — The PPS source. The Pulse-per-second signal is responsible for incrementing the timestamp UTC, and can be obtained from the MUX6 divider output (also *DBUS6*), from an external source (rear panel PPS input), or from the internal oscillator.

0 Idle.

1 DBUS (MUX6).

2 External Source.

3 Internal.

**Firmware Version Register [62]**

RegA

RegB

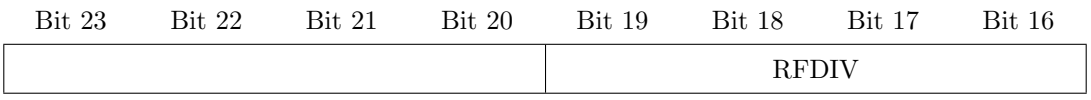

RegC

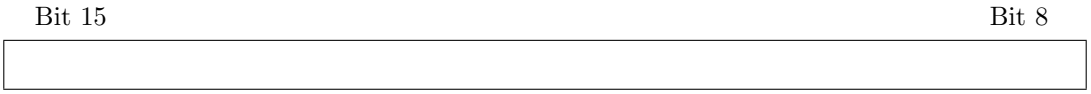

FRMVERSION The STD-EVO current firmware version, which is represented by the first 12 characters of the firmware commit hash.

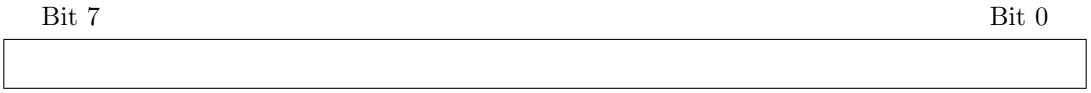
**Configuration Register [63]**

RegA

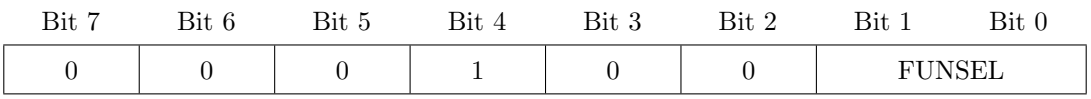

RegB




RegC



ALIVE — The alive counter is incremented by the internal oscillator. It starts once the STD-EVO function is configured as EVG, EVR, or FOUT.

RFDIV — The RF divider. The divider to be applied to *RF IN* input signal. The output of the RF divider is the *event clock*, which is the parallel clock for data frame transmission in the Timing System, the *Event Sequencer* clock, and the *MUX* dividers input.

Prescaler = RFDIV + 1.

Example: For RFDIV = 0, RF input is divided by 1.

FUNSEL — STD-EVO function selection.

0 FOUT (default)

1 EVR

2 EVG

*RFDIV* MUST be set to a valid value, i.e., that keeps *event clock* within bounds, whenever the RF signal is recovered after a loss (*RFINS* equal *0*). If *RFDIV* is not reset after RF is recovered, even if the current divider value is valid, the module will remain in Error state.

## 3.2 STD-EVO/EVR

When configured as EVR, the STD-EVO is an *Event Receiver*. The role of an *Event Receiver* is to convert the data frames broadcast by the *Event Generator* (EVG) into trigger and clock signals that can be transmitted to devices in the accelerator. The STD-EVO module has three inputs in the front panel, two of which are used in the EVR configuration: *UPLINK*, and *ACIN INH*. The signal associated with each input is described below.

The *UPLINK* input is connected to one of the EVG's SFP outputs (generally with fiber optic cables). This input receives data frames broadcast by the EVG containing timing information to be converted into triggers and clocks.

The *INH* input (ACIN INH) is an interlock active-low input.

The Timing System data frame is composed of two parts of 8 bits each: the event code, which is converted by the EVR into a trigger, and the *Distributed Bus* (*DBUS*), which carries information of 8 different clocks (see figure 5). The parallel frequency for transmission of data frames by the EVG is defined by the EVG's *event clock*, which is a submultiple of the RF frequency. In addition to obtaining the event code and *Distributed Bus* clocks, the *Event Receiver* uses the received data frames to recover the *event clock* and lock itself to the EVG's reference clock.

| *Event code* | *DBus* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 8 bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Figure 5: Timing System Data Frame

The EVR has 8 SFP outputs in the front panel (OUT0 - OUT7). Each OUTx can be configured to output one of the *Distributed Bus* clocks, or transmit one of the configured triggers.

The EVR triggers are configured through 24 independent channels (OTP channels). Each OTP channel can be configured to generate a trigger in response to the reception of a given event. The OTP channel also has other parameters, which define the characteristics of the output trigger, i.e., delay, width, polarity, and number of pulses.

The EVR has 12 outputs in the rear panel, which are permanently mapped to the OTP0 - OTP11 outputs, respectively.

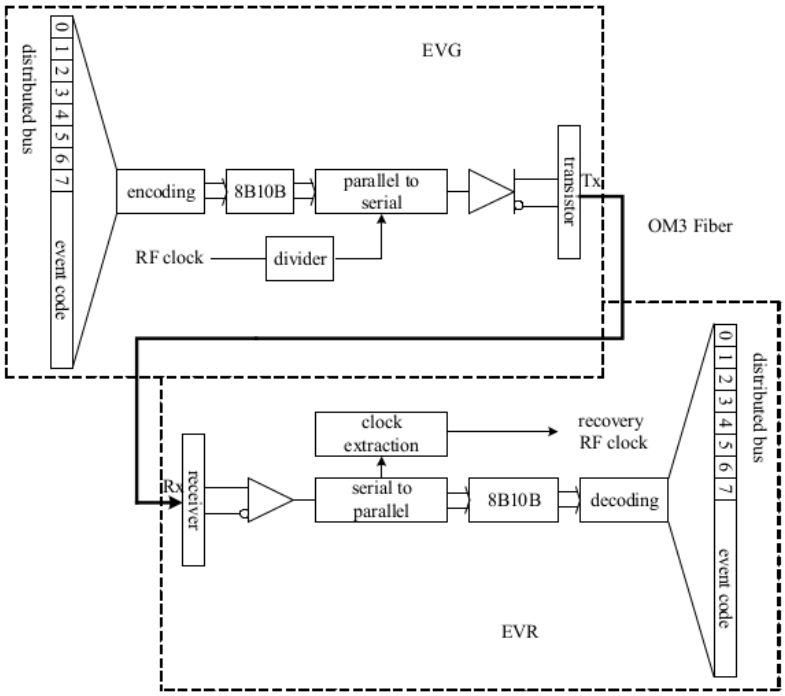
The *Event Receiver* has timestamping functinality. It has a timestamp register containing the current UTC timestamp, which is incremented through special event codes, or the *Distributed Bus* bit 6 (DBUS 6) clock, depending on the configuration. The timestamp register contains a subsecond counter as well. This counter is updated by the *event clock* recovered from the *UPLINK*. The module has also a timestamp FIFO for storing event codes and the timestamp in which they were received. The *OTP channels* have a parameter which tells whether the event being monitored should be stored upon reception.

Details of the EVR submodules are provided later in this section.

### 3.2.1 Clock Recovery

The *Event Generator* broadcasts the event code and *Distributed Bus* data frame every event clock period, even when there is no event scheduled, in which case a dummy event with code 0 is broadcast. The data frame is broadcast using 8b10b encoding (see figure 6). The *Event Receivers* use the data frames that arrive from the *UPLINK* to recover the *event clock*.

Figure 6: Timing system encoding-decoding scheme




### 3.2.2 Trigger Generation

The EVR has 24 *OTP* channels, each of which can be configured to generate a trigger upon the reception of a given event. The OTP channels parameters are configured in the OTPx Register. The *EVT* parameter specifies the event code responsible for generating the trigger. The *DELAY* parameter

specifies the delay between the event reception and the output trigger in *event clock* period units. The *WIDTH* parameter specifies the width of the pulse generated in *event clock* period units. The *POL* parameter specifies the pulse polarity. The *PULSES* parameter specifies the number of pulses to be generated in response to the event reception. When *PULSES* is greater than one, the trigger generated is in fact a pulse train. The pulse train period is 2 times *WIDTH* and the duty cycle is 0.5.

The EVR's *Timestamp FIFO* can store the instant when some event is received. Each *OTP* channel has a *TIME* parameter that enables the timestamping function for the event code being monitored.

### 3.2.3 SFP Ouputs

The SFP outputs correspond to the OUT0 - OUT7 outputs in the EVR front panel. The OUT0 - OUT7 outputs can be configured in the OUTx Register. The *OUTx* signal source can be mapped to one of the *OTP* channels, or any of the *Distributed Bus* clocks. The source selection is defined by the *SEL* parameter.

The *ITL* parameter enables/disables the interlock function for the corresponding *OUT* channel. When the *OUTx* channel has ITL *enabled*, the channel is inhibited whenever the EVR's interlock input (*INH*) is active (active-low signal).

The OUTx signal has an associated RF delay (*RFDLY*) and fine delay (*FINEDLY*). The RF delay resolution is $\frac{1}{20}$ of *event clock* period. When the value of *RFDLY* is set to 31, the RF delay of the OUTx output is set by the EVG's *Event Sequencer* special event codes 0x40 - 0x53, which set the delay value from 0 to 19 respectively. The RF delay special event codes can be written to the EVG's *Sequence RAM*, in order to give event timestamps enough resolution to, for example, allow single bucket injection. The fine delay resolution is 5ps, and may be used for fine adjustment of the trigger timing.

### 3.2.4 POF Ouputs

There are 12 Plastic Optical Fiber (POF) outputs in the EVR rear panel. The POF outputs are permanently mapped to the OTP0 - OTP11 channels outputs.

### 3.2.5 Timestamp

The EVR has timestamping function, which allows it to register when events of interest are received. To this end, the module maintains a timestamp register containing the 64-bit Timing System timestamp, which comprises a 32-bit *UTC* timestamp and a 32-bit *SUBSECOND* timestamp. The EVG sets the EVR's timestamp (generally once), and then increments it every second. The timestamp settings and status can be accessed in the Timestamp Register.

The 32-bit *UTC* timestamp stores the number of seconds passed since some epoch. This value can only be modified by a timestamp broadcast. When the EVG broadcasts its UTC timestamp, the EVR first receives the special event code 0x74. The event 0x74 resets the *SUBSECOND* counter and tells the EVR to treat the next 4 events as UTC information, instead of actual event codes. The UTC information received then replaces the current 4-byte *UTC* value.

The *SUBSECOND* counter is incremented by the recovered *event clock*, providing *event clock* period resolution to the timestamp.

The *TIMESRC* field specifies the PPS source, i.e., the signal which is going to increment the EVR's UTC. When *TIMESRC* is configured as *EVENT*, the *UTC* field is incremented by the special event 0x73, which is sent at the start of every second. When the *TIMESRC* is configured as *DBUS*, the *UTC* field is incremented by the *DBUS 6* clock. When *TIMESRC* is *INTERNAL*, the internal oscillator provides the increment signal. Once a PPS is received, the *SUBSECOND* counter is reset and the *UTC* timestamp is incremented.

In addition to the timestamp special events already mentioned, there are also event code 0x71, which resets the *SUBSECOND* timestamp, and event code 0x72, which resets the *UTC* timestamp.

**Timestamp FIFO** The EVR has a *Timestamp FIFO*, also referred to as *Timestamp Log*, capable of storing 16384 sets of event code and timestamp (32-bit *UTC* and 32-bit *SUBSECOND*). The purpose of the *Timestamp Log* is to store the instant when relevant events are received. The Timestamp Log Register is used to read and command the *Timestamp FIFO*. Each OTP channel has a timestamping setting, defining whether the event code mapped to that channel should be stored along with its timestamp once received.

The *Timestamp FIFO* operates as a circular buffer, replacing the oldest timestamp value by the newest, in the case that the FIFO is full and a new timestamp is stored. The *EMPTY* and *FULL* flags indicate when the FIFO is empty, or full respectively.

Writing to the *PULL* field pulls the *Timestamp Log* oldest value. Reading *PULL* has no effect. The pull action updates the *Timestamp Log*'s *UTC*, *SUBSECOND*, and *EVENT* fields with the UTC, subsecond, and event code obtained from the *Timestamp FIFO*. The pulled information is erased from the Log. The procedure for reading the *Timestamp Log* consists of writing to *PULL* and reading the Log's *UTC*, *SUBSECOND*, and *EVENT* fields.

The *LOGCOUNT* field indicates how many sets of event code and timestamp are stored in the *Timestamp FIFO*. The *RSTLOG* field resets the *Timestamp FIFO*, i.e., clear all stored values. The *STOPLOG* field stops the timestamping function while set. Both *RSTLOG* and *STOPLOG* return their current value when read.

### 3.2.6 Registers

| Address | Register Name | Description |
|---|---|---|
| **0** | Control and Status Register | General settings and status. Enable/disable module, *UPLINK* status, and interlock input status. |
| **1-16 25-32** | OTPx Register | OTPx settings. Specifies OTPx trigger delay, width, polarity, number of pulses, and mapped event. Also enables/disables timestamping option. |
| **17-24** | OUTx Register | OUTx settings. Specifies OUTx output signal source, RF delay, and fine delay. Also enables/disables the interlock function for the channel being configured. |
| **51** | Timestamp Register | Timestamp information and settings. Define PPS signal source, reading of *UTC* and *SUBSECOND* values. |
| **52** | Timestamp Log Register | Timestamp Log settings. *Timestamp FIFO* reading and commanding. |
| **62** | Firmware Version Register | 12-byte code for current firmware version. |
| **63** | Configuration Register | Main STD-EVO configuration options. Function selection (EVG, EVR, FOUT), and *Clock mode* setting. |

**Control and Status Register [0]**

RegC

| Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 |
|---|---|---|---|---|---|---|---|
| LINK | INHS | | | | | | |

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| | | | | | | | EVREN |

EVREN — Enable/Disable EVR. Disabling the EVR disables all of its outputs.

- 0 Disable
- 1 Enable

INHS — Interlock input status. The interlock input status only affects the OUTx outpus whose interlock function is enabled.

- 0 Disserted
- 1 Asserted

LINK — *UPLINK* status.

- 0 Unlink
- 1 Link

### OTPx Register [1-16] [25-32]

OTP0 - OTP15: Address 1 - 16

OTP16 - OTP23: Address 25 - 32

#### RegA

| Bit 31 | Bit 30 | Bit 29 | Bit 28 | Bit 27 | Bit 26 | Bit 25 | Bit 24 |
|---|---|---|---|---|---|---|---|
| EN | POL | TIME | | | | | |

| Bit 23 | Bit 16 |
|---|---|
| PULSES | |

| Bit 15 | Bit 8 |
|---|---|
| PULSES | |

| Bit 7 | Bit 0 |
|---|---|
| EVT | |

#### RegB

| Bit 31 | Bit 0 |
|---|---|
| DELAY | |

#### RegC

| Bit 31 | Bit 0 |
|---|---|
| WIDTH | |

| | |
|---|---|
| EN | Enable/Disable OTPx. Disabling the OTPx channel does not disable its timestamping function.<br>0 Disable<br>1 Enable |
| POL | OTPx polarity. Specify the polarity of the OTPx output trigger.<br>0 Normal<br>1 Inverted |
| TIME | Enable/Disable OTPx timestamping. When enabled, once the mapped event is received, the corresponding code and timestamp are stored in the *Timestamp FIFO*.<br>0 Timestamping<br>1 Not timestamping |
| PULSES | OTPx number of pulses. Specify the number of pulses to be generated. When the number of pulses is greater than 1, the output is a pulse train with duty cycle 0.5. |
| EVT | OTPx mapped event. Specify the event to generate the trigger. |
| DELAY | OTPx delay. Specify the delay between the reception of the mapped event and the trigger output. |
| WIDTH | OTPx width. Specify the width of the trigger/pulse train pulse(s). |

**OUTx Register** **[17-24]**

RegA

| Bit 31 | Bit 30 | Bit 29 | Bit 28 | Bit 27 | Bit 26 | Bit 25 | Bit 24 |
|---|---|---|---|---|---|---|---|
| ITL | | | | | | | |

| Bit 23 | | | | | | | Bit 16 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| Bit 15 | | | | | | | Bit 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| | SEL | | | | | | |

RegB

| Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 |
|---|---|---|---|---|---|---|---|
| | | | | | | FINEDLY | |

| Bit 7 | | | | | | | Bit 0 |
|---|---|---|---|---|---|---|---|
| FINEDLY | | | | | | | |

RegC

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| | | | RFDLY | | | | |

ITL — Enable/Disable OUTx interlock function. When the interlock function is enabled, the OUTx will be inhibited while the EVR interlock input (*INH*) is active.

- 0 Disable
- 1 Enable

SEL — OUTx source selection. Specifies the source of the OUTx signal from one of the *OTP* channels or *Distributed Bus* clocks.

- 0x10 - 0x1F OTP0 - OTP15
- 0x20 - 0x27 Dbus0 - Dbus7
- 0x30 - 0x37 OTP16 - OTP23

FINEDLY — OUTx fine delay. The fine delay between the event code reception and the OUTx trigger. The fine delay unit is 5ps.

RFDLY — OUTx RF delay. The RF delay between the event code reception and the OUTx trigger. The RF delay unit is $\frac{1}{20}$ of *event clock* period. When *RFDLY* is set to 31, the delay is defined by *Event Sequencer* event codes 0x40 - 0x53, which set the RF delay to 0 - 19, respectively.

**Timestamp Register [51]**

RegA

| Bit 31 | Bit 0 |
|---|---|
| UTC | |

RegB

| Bit 31 | Bit 0 |
|---|---|
| SUBSECOND | |

RegC

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| | | | | | | TIMESRC | |

UTC — The timestamp UTC field, which is a 32-bit counter to store the number of seconds passed since some epoch. The counter is incremented by the source defined in *TIMESRC*. The UTC can only be modified by EVG timestamp broadcasts or PPS signal.

SUBSECOND — The timestamp subsecond field, which is a 32-bit counter to store the subsecond portion of the Timing System timestamp. SUBSECOND is incremented by the recovered *event clock*, and thus, it has *event clock* period resolution. The SUBSECOND is reset whenever a PPS signal or UTC broadcast is received.

TIMESRC — The Pulse-per-second signal source, which is responsible for incrementing the timestamp UTC. The PPS signal can be obtained from the clock transmitted by the *Distributed Bus* bit 6, from the special event code 0x73, which is broadcast by the EVG at the start of every second, or from the internal oscillator.

0 Idle

1 DBUS (DBUS6)

2 Event

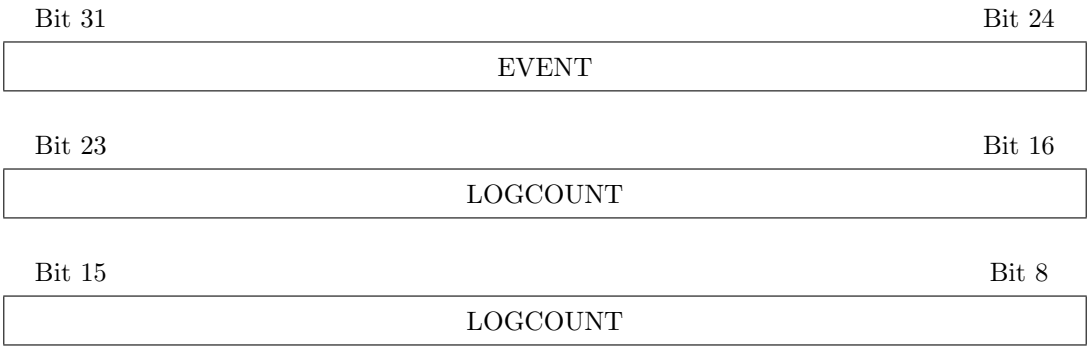
3 Internal

**Timestamp Log Register [52]**

RegA


RegB


RegC



| | |
|---|---|
| UTC | Last UTC timestamp pulled from the *Timestamp FIFO*. |
| SUBSECOND | Last subsecond timestamp pulled from the *Timestamp FIFO*. |
| EVENT | Last event code pulled from the *Timestamp FIFO*. |
| LOGCOUNT | Number of sets of event code and timestamp in the *Timestamp FIFO*. |
| STOPLOG | Stop log function. When enabled, the timestamping function is stopped.<br>0 Disable<br>1 Enable |
| RSTLOG | Reset log function. When enabled, the *Timestamp FIFO* is cleared.<br>0 Disable<br>1 Enable |
| PULL | Pull timestamp from *Timestamp FIFO*. Writing to *PULL* moves the oldest set of event code and timestamp from the *Timestamp FIFO* to the *UTC*, *SUBSECOND*, and *EVENT* fields of the *Timestamp Log Register*, where it is available for reading. |
| FULL | *Timestamp FIFO* full flag. The full flag is set while LOGCOUNT is equal to 16384. While *RSTLOG* is set, both *FULL* and *EMPTY* flags stay in 1.<br>0 FIFO is not full<br>1 FIFO is full |
| EMPTY | *Timestamp FIFO* empty. The empty flag is set while LOGCOUNT is 0. While *RSTLOG* is set, both *FULL* and *EMPTY* flags stay in 1.<br>0 FIFO is not empty<br>1 FIFO is empty |

When the *UPLINK* signal is lost, *STOPLOG* is automatically set to 1. In this circumstance, *STOPLOG* can only be disabled after a *Timestamp Log* reset (*RSTLOG* set to 1).

**Firmware Version Register [62]**

RegA


RegB


RegC

FRMVERSION The STD-EVO current firmware version, which is represented by the first 12 characters of the firmware commit hash.

**Configuration Register [63]**

RegA

| Bit 31 | ... | Bit 0 |
|---|---|---|
| ALIVE | | |

RegB

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| | | | CLKMODE | | | | |

RegC

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | FUNSEL | |

ALIVE The alive counter is incremented by the internal oscillator. It starts once the STD-EVO function is configured as EVG, EVR, or FOUT.

CLKMODE Clock mode. Set according to *UPLINK* event clock frequency.

- 11 60MHz - 62.5MHz
- 12 63MHz - 77MHz
- 13 77.5MHz - 91.5MHz
- 14 92MHz - 106MHz
- 15 106MHz - 120.5MHz
- 16 121MHz - 135MHz

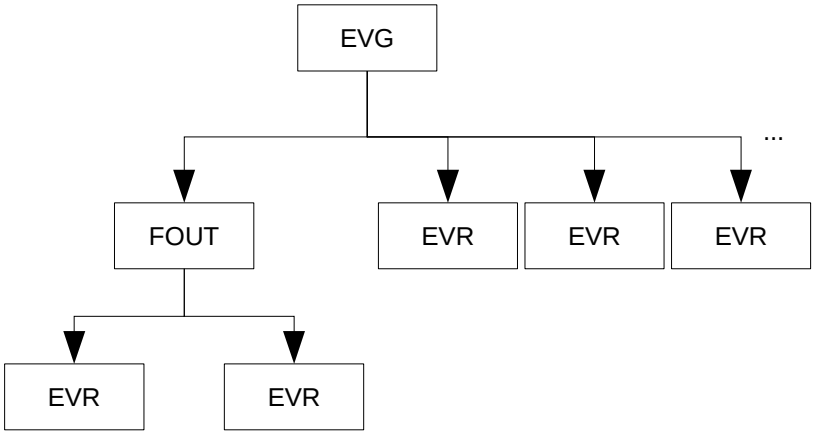
FUNSEL STD-EVO function selection.

- 0 FOUT (default)
- 1 EVR
- 2 EVG

## 3.3 STD-EVO/FOUT

When configured as FOUT, the STD-EVO is a *fanout*. The role of a *fanout* is to broadcast the information, i.e., data frames, received in its *UPLINK* to all of its SFP outputs (OUT0 - OUT7), extending the output capability of the *Event Generator* (see figure 7).

Figure 7: FOUT extending the EVG output




### 3.3.1 Registers

| Address | Register Name | Description |
|---|---|---|
| **0** | Control and Status Register | General settings and status. Enable/disable module, *UPLINK* status, and downlink status. |

**Control and Status Register [0]**

RegC

| Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 |
|---|---|---|---|---|---|---|---|
| LOS0 | LOS1 | LOS2 | LOS3 | LOS4 | LOS5 | LOS6 | LOS7 |

| Bit 23 | | | | | | | Bit 16 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 |
|---|---|---|---|---|---|---|---|
| LINK | | | | | | | |

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| | | | | | | | FOUTEN |

| | |
|---|---|
| FOUTEN | Enable/Disable FOUT. Disabling the FOUT disables all of its outputs. |
| | 0 Disable |
| | 1 Enable |
| LINK | *UPLINK* status. |
| | 0 Unlink |
| | 1 Link |
| LOS0 - LOS7 | Downlink status of OUT0 - OUT7 outputs, respectively. |
| | 0 Unlink |
| | 1 Link |

# 4 Ethernet Network Interface

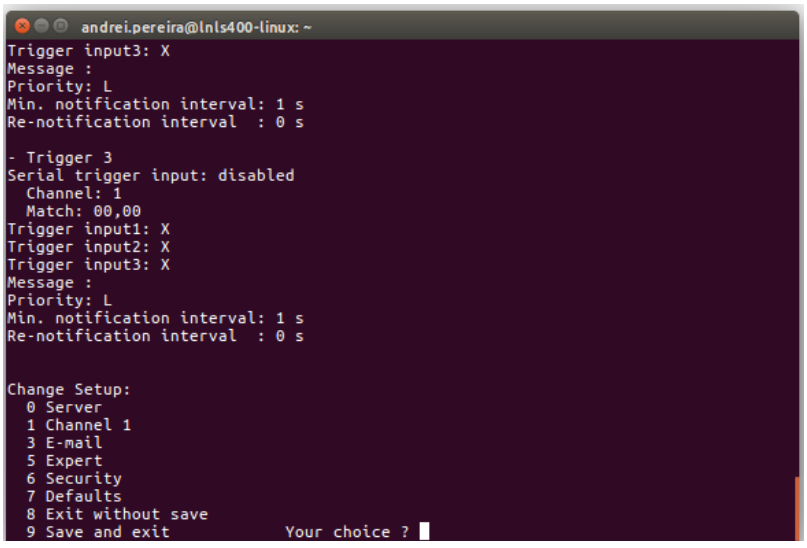
- 10/100Mbit Ethernet interface.
- UDP protocol by default.
- DHCP client by default.


4.1 Network configuration . . . . . 24
4.2 Register Read/Write . . . . . 25


## 4.1 Network configuration

In order to modify the module's network configurations, e.g., the IP address, the Telnet program command *telnet <IP address> 9999* can be used. Another option is to use the web browser, and type the module's IP address directly into the address bar, which will open the configuration page.

Figure 8: Setup using Telnet



## 4.2 Register Read/Write

Both read and write operations use the same structure for the UDP data frame, which is the following:

| Byte 0 | Byte 1 – Byte 4 | Byte 5 – Byte 8 | Byte 9 – Byte 12 |
|---|---|---|---|
| Command | RegA | RegB | RegC |

The *RegA*, *RegB*, and *RegC* sections in the UDP data frame correspond to the 32-bit register sections of same name in each register. These sections carry the information to be written/read. The operation selection and register are specified by the *Command* byte of the UDP data frame.

**Read (request)** In order to read a register, the UDP data frame *Command* byte must agree with the following rule:

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | Address | | | | | |

The first bits, from left to right, must be 1 and 0 respectively, followed by the register address.

**Read (response)** The response to a read request has a different *Command* byte, which is represented below:

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Address | | | | | |

The first bits, from left to right, must be 1 and 1 respectively, followed by the register address.

**Write** In order to write to a register, the UDP data frame *Command* byte must agree with the following rule:

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | Address | | | | | |

The first bits, from left to right, must be 0 and 1 respectively, followed by the register address.

# A Special Events

| Event code | Description |
|---|---|
| *General Purpose*[1] | |
| 0x00 | Idle Event |
| *Sequence RAM commanding* | |
| 0x70 | SEQRAM stop and wait a trigger to continue |
| 0x7E | SEQRAM switch |
| 0x7F | End of sequence, back to top |
| *Timestamp*[1] | |
| 0x71 | Reset subsecond timestamp register |
| 0x72 | Reset UTC timestamp register |
| 0x73 | Increment by 1 the UTC timestamp register and reset subsecond |
| 0x74 | Start of 32-bit UTC timestamp register update |
| *OUTx RF delay setting* | |
| 0x40 | delay = 0 |
| 0x41 | delay = 1 |
| 0x42 | delay = 2 |
| 0x43 | delay = 3 |
| 0x44 | delay = 4 |
| 0x45 | delay = 5 |
| 0x46 | delay = 6 |
| 0x47 | delay = 7 |
| 0x48 | delay = 8 |
| 0x49 | delay = 9 |
| 0x4A | delay = 10 |
| 0x4B | delay = 11 |
| 0x4C | delay = 12 |
| 0x4D | delay = 13 |
| 0x4E | delay = 14 |
| 0x4F | delay = 15 |
| 0x50 | delay = 16 |
| 0x51 | delay = 17 |
| 0x52 | delay = 18 |
| 0x53 | delay = 19 |

[1]Generated by module regardless of any configuration.