EVR Manual

Contents

1 Introduction

Event receiver is a part of a timing system. A timing system consists of an event generator (EVG), a series of event receivers (EVR), software controlling them and a timing network. EVG generates a series of events, which are delivered to EVRs through a timing network. An EVR is then configured to respond to specific events in various ways, including processing EPICS records and generating pulses, synchronized clock or custom signals on its outputs.

mrfioc2 [9] is an EPICS device support for the Micro Research Finland(MRF) [7] timing system. The mrfioc2 enables us to configure and use the event generators and event receivers in the timing system. It comprises of EPICS device support for MRF timing system and uses devlib2 [1] with additional kernel modules (eg. PCIe) for communication with the hardware.

The rest of this document describes the mrfioc2 device support in regards to the Event Receiver. EVR and its configuration is described, then instructions for starting a new IOC application for EVR are provided. The document continues with basic developer information for the mrfioc2 and ends with a short description of the EVR GUI. Some sections deal with PSI specifics. These are mostly simplifications in the deployment or build process for the mrfioc2 device support.

2 Event Receiver

EVRs are available in various form factors, each supporting the same basic functionality (executing functions, manipulating DBus,...), but with different number of components (inputs, outputs,...). Each of the EVR components and functionalities is configurable by the user and is briefly described in the following sections. It is possible to configure the EVR through the GUI (described in Section 8) at runtime, or by setting the appropriate macros in the EVR substitution file[1] in your IOC application.

How to read this section: Each sub-section starts with a short description of an EVR component or functionality. It is followed by the description of available macros, that can be used in a substitution file to configure the specific component or functionality of the EVR. Macro name, its default value, a description, available settings with values or other important information regarding the configuration is listed.

> Example: **macro name**=*default value* : Description, with available settings. `Setting name (macro value)` or `important information` is emphasized.

The exact number of form factor specific components is available in [6].

2.1 General configuration

There is usually only one event receiver controlled from a single IOC application, though it is possible to configure more. Thus the name of the EVR and the system name it belongs to must be defined.

Substitution file macros

- **SYS**=*MTEST-VME-EVRTEST* : The system name.
- **EVR**=*EVR0* : The name of the connected Event Receiver, which should be the same as defined in the startup script.

For EVR form factors that support GTX outputs (more about these is available in [6]), there is additional configuration option:

[1]On the PSI infrastructure, a prepared substitution file for VME-EVR-230RF form factor `mrfioc2/PSI/evr_ex.subs` with all the macros and short documentation is available (file list item 6 in Section 5.2)

- **ExtInhib-Sel**=*0* : This macro takes effect only when configuring GTX ouputs on cPCI-EVRTG-300 form factor. Either honor the hardware inhibit signal(`0`) or don't care about hardware inhibit input state(`1`).

2.2 Distributed Bus (DBus)

The distributed bus is able to carry 8 signals, that are propagated throughout the timing network with the event clock rate. Individual DBus signals (bits) can be outputted through programmable outputs(FrontOut, FrontUnivOut, RearUniv), or used as inputs (described in Section 2.6).

2.3 Events and event clock

EVR receives events transmitted by an event generator from the timing network. Event clock is the frequency, at which the events are transmitted. All the EVRs lock to the phase and frequency of the event clock, thus it is synchronized across the entire timing system.

Substitution file macros

- **Link-Clk-SP**=*124.916* : The event receiver requires a reference clock to be able to synchronize on the incoming event stream sent by the event generator. For flexibility, a programmable reference clock is provided to allow the use of the equipment in various applications with varying frequency requirements. It must be close enough to the EVG clock to allow phase locking with EVR. Available values are `50 MHz - 150 MHz`.

- **Link-RxMode-Sel**=*1* : Set whether the `DBus(0)` or `DBus + Data Buffer(1)` are sent downstream (to the timing network). Distributed bus bandwidth may be shared by transmission of a configurable size data buffer to up to 2 kbytes. When data transmission is enabled the distributed bus bandwidth is halved.

2.4 Pulse Generator - Pulser (Pul)

A pulse generator is able to output a configured pulse upon reception of an event.

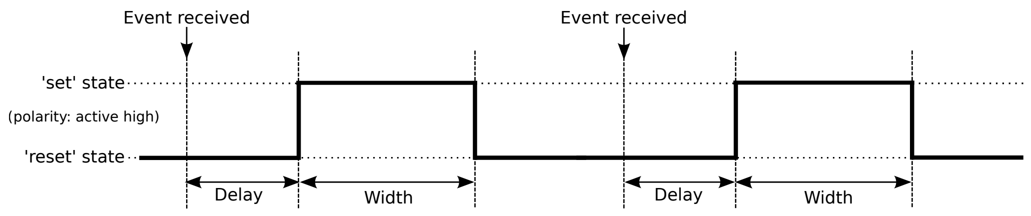

Figure 1: An example of a pulse generated after the reception of the event.

Figure 1 shows how the pulse is generated by switching between set and reset states, where the states determine the logic level of the pulser output signal.

The pulser has configurable properties:

- **Polarity** determines the logic level of the `set` and `reset` states. When polarity is set to
  - `active high`, the `set state` puts the pulser output to logic high and the `reset state` puts the pulser output to logic low.
  - `active low`, the `set state` puts the pulser output to logic low and the `reset state` puts the pulser output to logic high.
- **Delay** determines the time from when the event was received to when the pulser enters the *set state*.
- **Width** determines the time from when the pulser enters the *set state* to when it enters the *reset state*.

Substitution file macros There are a maximum of 16 pulsers available, depending on the EVR form factor. Pulsers are named Pul#, where # is the ID of the pulser ranging, from 0 to 15.

- **Pul#-Ena-Sel**=*1* When `disabled(0)`, the output of the pulser will remain in its reset state. The pulser must be `enabled(1)` when used.
- **Pul#-Polarity-Sel**=*0* : Sets the pulser **polarity** to `active high(0)` or `active low(1)`.
- **Pul#-Delay-SP**=*0* : Sets the pulse **delay** in range of `0 us - 4294967295 us`.
- **Pul#-Width-SP**=*0* : Sets the pulse **width** in range of `0 us- 65535 us`.

- **Pul#-Prescaler-SP**=*1* : Decreases the resolution of both **delay** and **width** by an integer factor. Value range: `0-255`. Pulsers without prescalers use a fixed prescale value of 1.

Each received event can activate a function of the pulser:

- **Trig** function uses polarity, delay and width to generate a pulse. Pulser starts in `reset state`. After the reception of an event, pulser waits `delay` ns and goes to `set state`. Then it waits for `width` ns and goes back to `reset state`.
- **Set** function puts the pulser to `set state`. Delay and width properties are ignored.
- **Reset** function puts the pulser to the `reset state`. Delay and width properties are ignored.

Substitution file macros for mapping events to pulser function are available in Section 2.11.3, together with an **example**.

2.5 Prescaler (PS)

Prescalers can be configured to output the event clock divided by an integer factor. There is a special event 123 (0x7b) that resets all the prescalers, so that the prescaled signal is in the same phase across all EVRs.

Substitution file macros There are a maximum of 4 prescalers available, depending on the EVR form factor. Prescalers are named PS#, where # is the ID of the prescaler, ranging from 0 to 3.

- **PS#-Div-SP**=*2* : Sets the integer divisor between the event clock and the prescaled event clock output in a range of `2-65535`.

2.6 Front Panel TTL Input (FPIn)

Front panel inputs can also be called External Event Inputs, because they can be configured to cause an event. The event can be local to the EVR (trigger mode) or sent through the timing network (backwards mode) when a condition occurs. The conditions are configured to respond to the front panel input signal logic level (level condition) or input signal edge (edge condition). Events generated by the front panel input logic are handled as any other events. Front panel inputs also provide configuration options for DBus signal manipulation.

Substitution file macros There are a maximum of 2 front panel TTL inputs available, depending on the EVR form factor. Front panel TTL inputs are named FPIn#, where # is the ID of the front panel TTL input, ranging from 0 to 1.

- **FPIn#-Lvl-Sel**=*1* : Determines if event is sent when the input signal logic level is low `Active Low(0)` or high `Active High(1)`, when using the `level` condition.
- **FPIn#-Edge-Sel**=*1* : Determines if event is sent on the falling `Active Falling(0)` or rising `Active Rising(1)` edge of the input signal, when using the `edge` condition.
- **FPIn#-Trig-Ext-Sel**=*0* : Selects the condition (`Off(0)`, `Level(1)` or `Edge(2)`) on which to trigger an event local to the EVR. This is the trigger mode condition.
- **FPIn#-Code-Ext-SP**=*0* : Sets the event which will be sent to timing network, whenever the trigger mode condition is met. Any event in range of `0-255` can be selected.
- **FPIn#-Trig-Back-Sel**=*0* : Selects the condition (`Off(0)`, `Level(1)` or `Edge(2)`) in which to send an event to the timing network. This is the backward mode condition.
- **FPIn#-Code-Back-SP**=*0* : Sets the event which will be sent to timing network, whenever the backwards mode condition is met. Any event in range of `0-255` can be selected.
- **FPIn#-DBus-Sel**=*0* : Set the upstream DBus bit mask which is driven by this input. DBus bits and the macro value are condensed with a bit-wise OR. Available values for the DBus bit mask are: `1 = Bit 0, 2 = Bit 1, 4 = Bit 2, 8 = Bit 3, 16 = Bit 4, 32 = Bit 5, 64 = Bit 6, 128 = Bit 7`.

2.7 Outputs

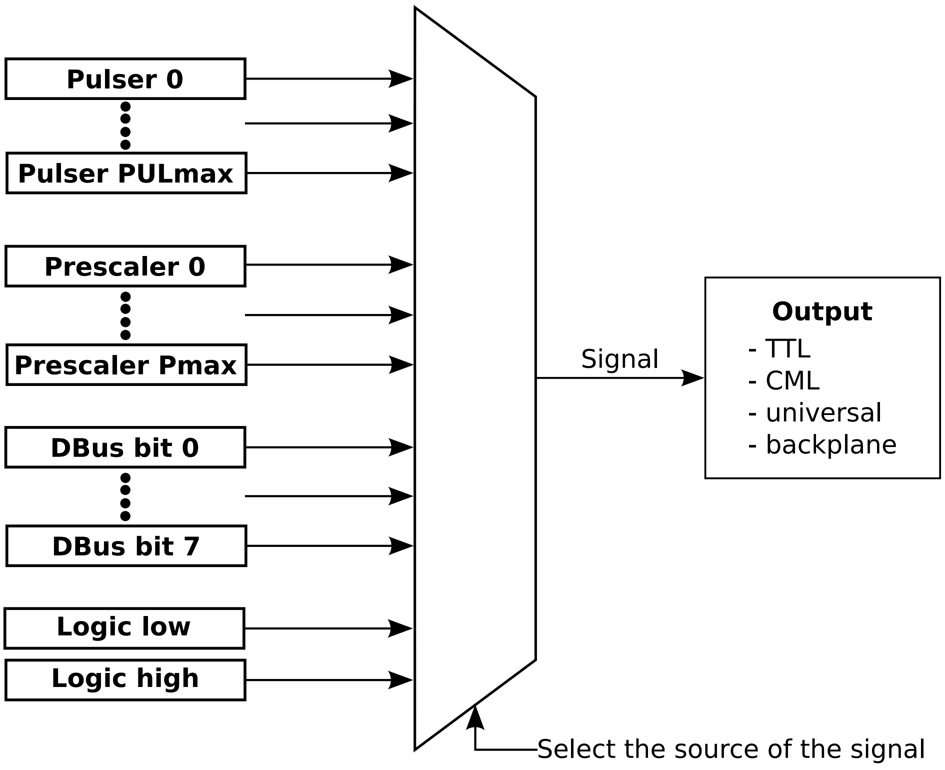

Figure 2: Available source signals for each output

An EVR has a number of outputs, each with a configurable source signal. As seen in Figure 2, any of the available pulser, prescaler, DBus, logic low or logic high signals can be selected as the output source signal. Depending on the output type(TTL, CML, ...), the signal can be send through directly, further manipulated or used as a trigger for a special function of that output.

Substitution file macros common to all outputs: Outputs, and the corresponding macro name prefixes, are named either *FrontOut#*, *FrontOutUniv#* or *RearUniv#*, where the range of the number # depends on the EVR form factor. Macros for the *FrontOut0* are described below.

- **FrontOut0-Ena-SP**=*1* : When set to `enabled(1)` the mapping defined in `FrontOut0-Src-SP` is used. When `disabled(0)`, a mapping of `Force Low(63)` from Table 1 is used.

Table 1: Output source signal mappings

mapping	**output source**
0	Pulser 0
1	Pulser 1
2	Pulser 2
3	Pulser 3
4	Pulser 4
5	Pulser 5
6	Pulser 6
7	Pulser 7
8	Pulser 8
9	Pulser 9
10	Pulser 10
11	Pulser 11
12	Pulser 12
13	Pulser 13
14	Pulser 14
15	Pulser 15
32	Distributed bus bit 0
33	Distributed bus bit 1
34	Distributed bus bit 2
35	Distributed bus bit 3
36	Distributed bus bit 4
37	Distributed bus bit 5
38	Distributed bus bit 6
39	Distributed bus bit 7
40	Prescaler 0
41	Prescaler 1
42	Prescaler 2
62	Logic High
63	Logic low

- **FrontOut0-Src-SP**=*63* : `Mappings` from the Table 1 are set here. Any of the available pulser, prescaler, DBus, logic low or logic high signals can be selected as the output source signal.

2.7.1 Front Panel TTL Output (FrontOut)

These outputs are capable of driving TTL compatible logic level signals. The output source signal is converted to TTL logic levels(LVTTL of max 3.3 Volts) and send through the output, as seen in Figure 3.

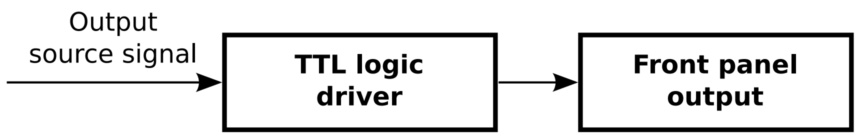

Figure 3: Front panel TTL output

2.7.2 Front Panel CML Output (FrontOut)

These outputs are capable of driving Current-mode logic compatible signals. Based on the output source signal and the selected CML mode, various patterns can be generated. As seen in Figure 4, patterns are generated using one of the three configurable modes. They are sent out with a bit rate of 20 times the event clock rate, thus the outputs allow for producing fine grained adjustable pulses and clock frequencies.

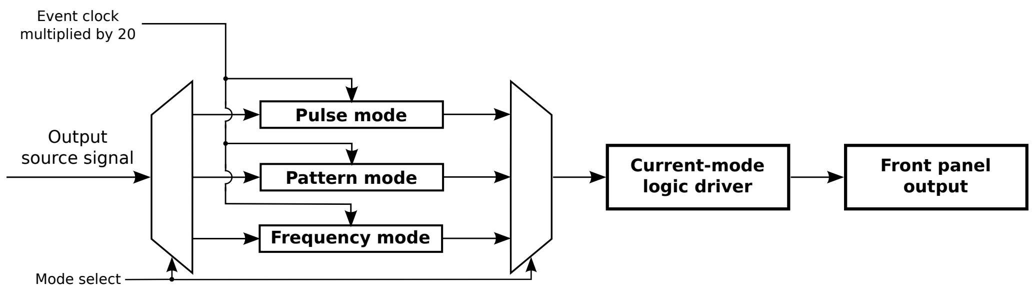

Figure 4: Overview of the front panel CML outputs

Substitution file macros for general CML configuration: There are a maximum of 8 CML outputs available, depending on the EVR form factor. CML outputs are named CML#, where # is the ID of the CML output, ranging from 0 to 7. Some form factors do not have any CML outputs.

- **CML#-Ena-Sel=** *0* : Output is `disabled(0)` or `enabled(1)`.
- **CML#-Pwr-Sel=***0* : Outputs can be `powered down(0)` or in normal `operation(1)`.

- **CML#-Rst-Sel**=*0* : It is possible to `reset CML output(1)` or leave it in `normal operation(0)`.
- **CML#-Mode-Sel**=*0* : There are three configurable modes, `pattern mode(0)`, `frequency mode(1)` and `waveform mode(2)`.

Patterns are defined by one of the configurable CML modes:

- In **pulse mode** a user configurable 20-bit pattern ia sent out based on the output source signal:
  - **rising edge**: pattern is sent out after the rising edge is detected and will interrupt the current pattern being sent.
  - **falling edge**: pattern is sent out after the falling edge is detected and will interrupt the current pattern being sent.
  - **logic high**: pattern is sent out after the rising edge is detected and will repeat until the falling edge.
  - **logic low**: pattern is sent out after the falling edge is detected and will repeat until the rising edge.
- In **frequency mode** a clock signal can be generated. Its frequency is tuned in steps of 20 times the event clock frequency. The clock signal is synchronized by the output source signal(pulser, DBus signal, ...).

  Frequency generation is triggered by a rising edge of the output source signal. After the trigger, it waits for a configured initial delay before it starts to generate a clock signal.

  Substitution file macros

  - **CML#-Freq:Lvl-SP**=*0* : set the starting logic level for the frequency generation. Options are:
    - `Active high(0)`: starting logic level is logic high
    - `Active low(1)`: starting logic level is logic low
  - **CML#-Freq:Init-SP**=*0* : Set the initial delay between when the frequency generation is triggered and when it starts to generate a clock signal. This allows for a phase difference between the output source signal and the output signal. The value range in ns depends on the event clock. It must be in range of 1 event clock period to 3276 event clock periods.

  - **CML#-Freq:High-SP**=*10* : sets the amount of time in ns the signal stays on logic high, before switching to logic low. The value range in ns depends on the event clock. It must be in range of 1 event clock period to 3276 event clock periods.
  - **CML#-Freq:Low-SP**=*10* : sets the amount of time in ns the signal stays logic low, before switching to logic high. The value range in ns depends on the event clock. It must be in range of 1 event clock period to 3276 event clock periods.

- In **pattern mode**, one can generate arbitrary bit patterns(waveforms), triggered by the output source signal(pulser, DBus signal,...). The waveform length is in multiple of 20 bits (each bit is 1/20 of the event clock period), where maximum waveform length is 20x2048 bits.

Substitution file macros

  - **CML#-Pat:WfCycle-SP**=*0* : In `single shot mode(0)` the waveform is sent only once per received trigger, where in `loop mode(1)` the pattern will continuously loop after the first trigger occurred.

There are two additional calculators available to simplify waveform creation.

A delay calculator has configurable `delay` and `width` macros to generate a pulse(similar to a pulser):

  - **CML#-WfCalc:Ena-SP**=*0* : `enable(1)` or `disable(0)` the calculation.
  - **CML#-WfCalc:Delay-SP**=*16* : can be used to specify the time in ns between when the mode is triggered and the pulse is generated. The value range in ns depends on the event clock. It must be in range of 1 event clock period to 2048 event clock periods.
  - **CML#-WfCalc:Width-SP**=*50* : can be used to specify the width of the pulse - the time the signal is stable at logic high. The value range in ns depends on the event clock. It must be in range of 1 event clock period to 2048 event clock periods.

A bunch train calculator that generates a series of consecutive pulses with width and delay fixed to (*eventclock*)/4. As seen in Figure 5, each bunch pattern consists of 5 logic low bits and 5 logic high bits. A bunch train always ends with 10 logic low bits.

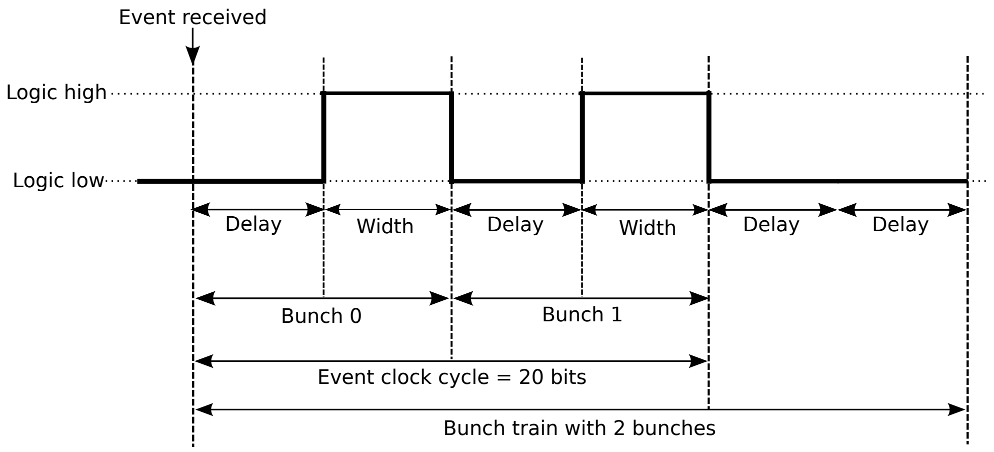

Figure 5: A bunch train example with 2 bunches

Bunch train calculator has the following substitution macros for configuration:

- **CML#-BunchTrain:Ena-SP**=*0* : `enable(1)` or `disable(0)` the bunch train calculator
- **CML#-BunchTrain:Size-SP**=*1* : set the number of bunches per train in range of `1-150`.

2.7.3 Front Panel Universal I/O (FrontUnivOut)

Universal I/O slots provide different types of input or output with exchangeable Universal I/O modules [7]. Each module provides two inputs or outputs (TTL, NIM or optical). The exact output signal is dependent on the inserted module. Figure 6, shows an example of a universal I/O module with two outputs inserted in front panel universal slot.

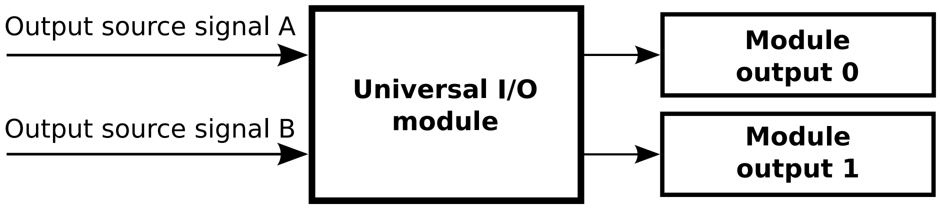

Figure 6: An example universal I/O module with two outputs.

2.7.4 Rear Universal output - Transition Board output (RearUniv)

An EVR has additional rear universal outputs. The exact output signal is dependent on the universal module inserted in the transition board.

2.8 UNIV-LVPECL-DLY high precision delay module

EVR has universal I/O slots, described in Section 2.7.3, where universal modules can be inserted. One of these is UNIV-LVPECL-DLY Delay module. This module will output a delayed signal from the appropriate FrontUnivOut output. The FrontUnivOut output source signal can be configured as described in Section 2.7 using mappings from Table 1. The delay can be configured from 2200 ps to 12430 ps, with 10 ps resolution.

Substitution file macros

- **SYS**=*MTEST-VME-EVRTEST* : The system name.
- **EVR**=*EVR0* : The name of the event receiver, where the module is inserted.
- **SLOT** : The universal slot of the EVR, where the module is inserted. FrontUnivOut0 is the first and FrontUnivOut1 is the second output of slot `0`, where FrontUnivOut2 is the first and FrontUnivOut3 is the second output of slot `1`.
- **Enabled**=*1* : `enable(1)` or `disable(0)` the module. When disabled, both outputs are pulled to logic low.
- **Delay0**=*2.2* : is a tunable delay for the first output. The value is in range of `2.2 ns - 12.43 ns`.
- **Delay1**=*2.2* : is a tunable delay for the second output. The value is in range of `2.2 ns - 12.43 ns`.

Example: Both front panel universal slots of the EVR0 are occupied since there are two delay modules inserted, one in slot 0 and one in slot 1. Module in slot 0 is enabled and has first output set to minimum delay and second output to maximum delay. Module in slot 1 is disabled with both output delays set to 5ns. The EVR belongs to the MTEST-VME-EVRTEST system.

```
file "evr-delayModule.template"
{
    { SYS     = MTEST-VME-EVRTEST,
      EVR     = EVR0,
      SLOT    = 0,

      Enabled = 1,
      Delay0  = 2.20,
      Delay1  = 12.43,
    }
    { SYS     = MTEST-VME-EVRTEST,
      EVR     = EVR0,
      SLOT    = 1,

      Enabled = 0,
      Delay0  = 5,
      Delay1  = 5,
    }
}
```

2.9 Time stamping mechanism

The functionality of the time stamping mechanism was not tested!

The event receiver supports a global time stamping mechanism. Time-stamp consists of a 32-bit time-stamp event counter and a 32-bit seconds counter. The seconds counter can be updated from the shift register. Counters are clocked using prescaled event clock, DBus bit 4, or on each reception of the mapped event (`TS Tick` special function described in Section 2.11.2). If configured, events together with their time-stamp are stored in a FIFO memory.

- **Time-Src-Sel**=*0* : Determines what causes the timestamp event counter to increment.

  - The `event clock(0)` source will use a prescaled EVR reference clock.
  - The `mapped codes(1)` increment the counter whenever certain event arrives. Time-stamp counter event can be defined using a `TS Tick` special function described in Section 2.11.2.
  - `DBus bit 4(2)` will increment the counter on the low-to-high transition of the DBus bit 4.

- **Time-Clock-SP**=*0.0* : Specifies the rate at which the timestamp event counter will be incremented. This determines the resolution of all timestamps. Use **in conjunction with** the `Time-Src-Sel`.

  When the timestamp source(`Time-Src-Sel`) is set to `Event clock` this macro value is used to prescale the EVR's reference clock frequency to the given frequency. Since this may not be exact it is recommended to read back the actual divider setting via the timestamp prescaler (`$(SYS)-$(EVR):Time-Div-I` record). In all modes this value is stored in memory and used to convert the timestamp event counter values from ticks to seconds. The units are in `MHz`.

2.10 Heartbeat monitor

A heartbeat monitor is provided to receive heartbeat events from the timing network. If no heartbeat event is received (in approx. 1.6 s), the heartbeat timeout occurs and a heartbeat flag is set.

2.11 Event mapping

As mentioned in Section 1, an EVR responds to events. In order for the EVR to respond, an appropriate mapping between the event and EVR function/component needs to be configured.

2.11.1 Trigger an EPICS event

Provides us with the ability to map between EPICS event (software) and an event from the timing system (hardware).

Substitution file macros:

- **SYS**=*MTEST-VME-EVRTEST* : The system name.

- **EVR**=*EVR0* : The name of the connected Event Receiver, which should be the same as defined in the startup script.
- **EVT** represents the event from the timing system. Set EVT=0 to disable.
- **CODE** represents EPICS event number (software).

Example: The configuration applies to EVR0 in the MTEST-VME-EVRTEST system. EPICS event 1 is set to trigger when an event 1 from the timing system is received, and EPICS event 2 to trigger on reception of event 2 from the timing system. It is `recommended to use the same EPICS event and timing system event numbers`, but it is not mandatory.

```
file "evr-softEvent.template"{
pattern { SYS,                  EVR,    EVT,    CODE}
        {"MTEST-VME-EVRTEST",   "EVR0", "1",    "1"}
        {"MTEST-VME-EVRTEST",   "EVR0", "2",    "2"}
}
```

It is also possible to manually forward link an appropriate event record instead of using EPICS events. The functionality of EPIC events and links is out of the scope of this documents.

2.11.2 Special functions

There is a number of special functions available, that can be activated on specified event. Each event can be mapped only to one function. There exist some default events, that always trigger specific function.

EVR functions:

- **Blink** : An LED on the EVRs front panel will blink when the event is received.
- **Forward** : The received event will be immediately retransmitted on the upstream event link.
- **Stop Log** : Freeze the circular event log buffer which contains up to 512 events with time-stamp information. A CPU interrupt will be raised which will cause this buffer to be downloaded. This might be a useful action to map to a fault event. (default event 121)

- **Log** : Include this event code in the circular event log.
- **Heartbeat** : This event resets the heartbeat timeout. See Section 2.10 for details. (default event 122)
- **Reset PS** : Resets the phase of all prescalers. (default event 123)
- **TS reset** : Loads the time-stamp seconds counter from the shift register and zeros the time-stamp event counter. See Section 2.9 for details. (default event 124)
- **TS tick** : When the time-stamp source is 'Mapped code' then any event with this mapping will cause the time-stamp event counter to increment. See Section 2.9 for details. (default event 125)
- **Shift 0** : Shifts the current value of the shift register up by one bit and sets the low bit to 0. See Section 2.9 for details. (default event 112)
- **Shift 1** : Shifts the current value of the shift register up by one bit and sets the low bit to 1. See Section 2.9 for details. (default event 113)
- **FIFO** : Bypass the automatic allocation mechanism and always include this code in the FIFO memory. See Section 2.9 for details.

Substitution file macros:

- **SYS**=*MTEST-VME-EVRTEST* : The system name.
- **EVR**=*EVR0* : The name of the connected event receiver, which should be the same as defined in the startup script.
- **EVT** represents an event from the timing system.
- **FUNC** represents one of the functions listed above.

Example: The following macros configure the event mappings to blink the LED on each occurrence of event 1, and event 6. The configuration applies to EVR0 in the MTEST-VME-EVRTEST system.

```
file "evr-specialFunctionMap.template"{
pattern { SYS,                    EVR,    EVT,   FUNC }
        {"MTEST-VME-EVRTEST",   "EVR0",  "1",   "Blink"}
        {"MTEST-VME-EVRTEST",   "EVR0",  "6",   "Blink"}
}
```

2.11.3 Pulser mapping

Event receivers have multiple pulsers that can preform several functions upon reception of events (described in Section 2.4). Each pulser-function combination can be mapped to multiple events.

Substitution file macros:

- **SYS**=*MTEST-VME-EVRTEST* : The system name.
- **EVR**=*EVR0* : The name of the connected event receiver, which should be the same as defined in the startup script.
- **PID** represents the pulser number
- **F** represents one of the functions (Trig, Set, Reset) described in Section 2.4.
- **EVT** represents an event from the timing network.
- **ID** Mappings must have unique ID for each PID-F combination.

Example: Pulser 0 is set to trigger on event 2 and event 3. It will also reset on event 4. The configuration applies to EVR0 in the MTEST-VME-EVRTEST system.

```
file "evr-pulserMap.template"{
pattern { SYS,                    EVR,    PID   F,      EVT, ID }
        {"MTEST-VME-EVRTEST",   "EVR0", 0,    Trig,    2,   0 }
        {"MTEST-VME-EVRTEST",   "EVR0", 0,    Trig,    3,   1 }
        {"MTEST-VME-EVRTEST",   "EVR0", 0,    Reset,   4,   0 }
}
```

3 PSI IOC startup

Section 2 of the Tutorial [4] describes a simple way of using the predefined templates with a generic startup script to create an IOC application for EVR. This Section describes how to use the template files directly from the mrfioc2 reporsitory [11] with a custom startup script.

The templates are available in `mrfioc2/evrMrmApp/Db/PSI` folder (folder list item 3 in Section 5.2). The following steps should be taken in order to create a new SWIT compliant IOC application for the Event Receiver:

1. Create a project folder, eg. `MTEST-VME-EVRTEST`. Let $< TOP >$ be the path to this folder, eg $< TOP >$ = `~/public/F/TEST/MTEST-VME-EVRTEST`.

   ```
   mkdir -p <TOP>
   ```

2. **Assuming** the form factor is VME-EVR-230RF the template is available at this location: `mrfioc2/evrMrmApp/Db/PSI/evr-vmerf230.template`. A form factor specific template can also be manually generated as described in Section 7.

3. Copy the template files to your project by issuing the following commands in the `mrfioc2/evrMrmApp/Db/PSI/` folder:

   ```
   cp evr-softEvent.template <top>/
   cp evr-pulserMap.template <top>/
   cp evr-specialFunctionMap.template <top>/
   cp evr-vmerf230.template <top>/
   cp evr-delayModule.template <top>/
   ```

4. Copy the substitution file to your project by issuing the following commands in the `mrfioc2/PSI/` folder:

   ```
   cp evr_ex.subs <top>/MTEST-VME-EVRTEST_EVR.subs
   ```

5. Create a startup script $< TOP >$`/MTEST-VME-EVRTEST_startup.script`, with the following content:

   ```
   ## Load IFC1210 devLib and pev modules
   require 'pev'
   ## Load mrfioc2 device support
   require 'mrfioc2'
   ```

```
#########################
#-----! EVR Setup ------!#
#########################
## Configure EVR
## Arguments:
##  - device name
##  - slot number
##  - A24 base address
##  - IRQ level
##  - IRQ vector

mrmEvrSetupVME(EVR0,2,0x3000000,4,0x28);
var dbTemplateMaxVars 500

## EVR init done
```

In this example the EVR named "EVR0" is present in VME slot 2. It is given a base A24 address of 0x3000000 and configured to interrupt on level 4 with interrupt vector 0x28. User should change the parameters of *mrmEvrSetupVME(EVR0, 2, 0x3000000, 4, 0x28);* according to his hardware setup.

Unused address range should be selected by the user based on resources available on VME. The user must be confident that the address space is available and free, since the system does not preform any checks.

However, the system checks if an EVR is available in the selected slot. If it is not found, the EVR setup will abort.

Command `var dbTemplateMaxVars 500` sets the upper limit for the number of macros processed in the substitution file.

6. Add `SYS` and `EVR` macros and remove `$(TEMPLATE_DIR)/` from each template in the substitution file. SYS is the system name, EVR is the name of the Event Receiver (as defined in the startup script) and `$(TEMPLATE_DIR)/` is the location of the predefined template files.

   Example: Original substitution file snippet:

   ...

```
file "$(TEMPLATE_DIR)/evr-softEvent.template"{
pattern { EVT,  CODE }
        { "1",    "1"}
        { "2",    "2"}
        { "3",    "3"}
}
...
```

Macros SYS and EVR added and `$(TEMPLATE_DIR)/` removed:

```
...
file "evr-softEvent.template"{
pattern { SYS,                  EVR,    EVT,    CODE }
        {"MTEST-VME-EVRTEST",   "EVR0", "1",    "1"}
        {"MTEST-VME-EVRTEST",   "EVR0", "2",    "2"}
        {"MTEST-VME-EVRTEST",   "EVR0", "3",    "3"}
}
...
```

7. Set the macro values in the substitution file according to your needs. Detailed description of the macros is available in Section 2. You can also follow the Tutorial [4] to get acquainted with the EVR basic functions and configuration.

8. Optionally, you can remove all the macros whose values you did not change.

9. Install the prepared IOC by running the command `swit -V` from your project folder $< TOP >$.

4 Standard IOC startup

The EPICS way.

5 mrfioc2 organization

Some important folders:

```
mrfioc2
    |-- configure
    |-- documentation
       |--PSI
    |-- evgMrmApp
        |-- Db
            |-- PSI
        |-- src
    |-- evrApp
        |-- Db
        |-- src
    |-- evrMrmApp
        |-- Db
            |-- PSI
        |-- src
    |-- iocBoot
    |-- mrfCommon
    |-- mrmShared
    |-- mrmtestApp
    |-- PSI
    |-- ui
        |-- EVG
        |-- EVR
```

5.1 Source code

- `evgMrmApp/src` contains source files for the an Event Generator.
- `evrApp/src` and `evrmrmApp/src` contains source files for an Event Receiver.
- `mrfCommon` contains common functions and definitions used in the mrfioc2 module.
- `mrfShared` contains code shared between EVG and EVR together with some OS specifics and kernel modules.
- `/` and `PSI` contain makefiles for building the mrfioc2 module.
- `configure` and `iocBoot` are standard IOC application folders.

5.2 PSI specifics

Some important folders:

1. `documentation/PSI` contains this document and a tutorial [4] with step-by-step instructions to configuring some of the basic functionalities of the Event Receiver, including both document sources.
2. `evgMrmApp/Db/PSI` contains database templates for an Event Generator.
3. `evrMrmApp/Db/PSI` contains database templates for an Event Receiver.
4. `PSI` contains substitution files used in creation of an IOC application and a makefile to build the mrfioc2 module using driver.makefile.
5. `ui` contains caQtDM screens for Event Receiver and Event Generator in `EVR` and `EVG` sub-folder, respectively.

Some important files:

1. `mrfioc2/evgMrmApp/Db/PSI/evg-vme.template` is an EVG form factor specific template file.
2. `mrfioc2/evgMrmApp/Db/PSI/evg-vme.substitutions` is an EVG form factor specific substitution file used to generate the form factor specific template file(list item 1).
3. `mrfioc2/PSI/evg_ex.subs` is the EVG substitution used in creation of an IOC application. It contains form factor specific macros for setting up EVG.
4. `mrfioc2/evrMrmApp/Db/PSI/evr-vmerf230.template` is an EVR form factor specific template file.
5. `mrfioc2/evrMrmApp/Db/PSI/evr-vmerf230.substitutions` is an EVR form factor specific substitution file used to generate the form factor specific template file(list item 4).
6. `mrfioc2/PSI/evr_ex.subs` is the EVR substitution used in creation of an IOC application. It contains form factor specific macros for setting up EVR.
7. `ui/EVG/startUI.sh` is a script used to launch the EVG GUI.
8. `ui/EVR/startUI.sh` is a script used to launch the EVR GUI.

6 Building from scratch

There are two possible ways to build the mrfioc2 device support. First is using the driver.makefile [10] on the PSI infrastructure, and second using the standard makefile.

Prerequisites

- EPICS Base 3.14.x [5].
- devLib2 [1].
- git [3], to clone the repository [11].
- optionally, MSI tool [8] for expanding databases.

6.1 PSI driver.makefile

To compile to a PSI style module, first clone the PSI git repository for mrfioc2:

```
git clone https://github.psi.ch/scm/ed/mrfioc2.git
```

and then run `make` in `mrfioc2/PSI` folder:

```
cd mrfioc2/PSI
make -f makefile
```

Compilation will only work on a PSI infrastructure where the correct applications and environment variables are already configured.

Template files should be created separately, following the instructions in Section 7.

6.2 Standard makefile

Git clone, edit `configure/CONFIG_SITE` and run ”make”.

7 Generating templates

As described in Section 2, the EVR has many form factors with variable number of components. In order to simplify creating IOC applications, a form factor specific template file is created (file list item 4 in Section 5).

To create such a template, use a form factor specific substitution file(eg. file list item 5) and expand it using the MSI tool [8]. The tool works the same way as the EPICS macro substitutions do. The output of the tool can then be used in the IOC application.

However, in order to set the default values of macros, while recursively expanding templates (using MSI), a trick needs to be used in the original substitution file.

Problem A record with a macro in VAL field might look like this:
`field( VAL , "$(MCR=2)")`. After using the MSI tool, the field would look like this:
`field( VAL , "2")`, which means the ability to set the macro value is lost. Or, if the macro is defined, the default value is lost.

Solution The record field should be encapsulated using nested macro, like so:
`field( VAL , "$($(OBJ)-Div-SP\=2)")` Here is what happens:

- `$(OBJ)` is extended when running the MSI tool. This is OK, since it contains template specific information.
- Notice that '=' is escaped. This means that the MSI tool will only unescape the default value, instead of applying it to the undefined macro.
- Using the MSI tool with macro `EVR=evr0, OBJ=evr0:PS0` yields

  `field( VAL , "$(evr0:PS0-Div-SP=2,undefined)")`
- The macro substitution `evr0:PS0-Div-SP=val` can then be used in our application substitution file to set the field value, otherwise it defaults to '2'.

  Described solution can be observed in `evr-Base.template`, `evr-Cml.template`, `evr-In.template`, `evr-Pulser.template` and `evr-Scale.template` located in `mrfioc2/evrMrmApp/Db/PSI/` folder.

7.1 Manually generating EVR template

A MSI tool [8] is required to generate a form factor specific template. It is recommended to use a predefined template, but you can create one yourself. To create a VME-EVR-230RF form factor compatible template file issue the following command in `mrfioc2/evrMrmApp/Db/PSI` folder:

```
msi -S evr-vmerf230.substitutions > evr-vmerf230.template
```

The command will use the EVR form factor specific `evr-vmerf230.substitutions` file to output the template to be used in your application.

8 GUI

There is a caQtDM [2] GUI for the Event Generator and Event Receiver available in `mrfioc2/ui/EVG` and `mrfioc2/ui/EVR`, respectively. These folders contain a script for launching the GUI.

EVR GUI script usage:

```
./start_EVR.sh -s <system name> [options]
```

EVR GUI script options:

```
-r <event receiver name> ..... set the event receiver name
                               (default:EVR0).
-h                       ..... shows the options and usage
```

Example: Open the GUI for the event receiver `EVR0`,
using system name `MTEST-VME-EVRTEST`.

```
./start_EVR.sh -s MTEST-VME-EVRTEST
```

References

[1] devLib extensions for PCI and VME64x bus access. http://epics.hg.sourceforge.net/hgweb/epics/devlib2.

[2] caQtDM - a medm replacement based on QT. http://epics.web.psi.ch/software/caqtdm/.

[3] GIT. http://git-scm.com/.

[4] Tutorial. https://github.psi.ch/projects/ED/repos/mrfioc2/browse/documentation/PSI/tutorial.pdf.

[5] EPICS base. http://www.aps.anl.gov/epics/base/R3-14/index.php.

[6] Micro Research Finland. Event Receiver Modular Register Map Manual (all form factors). http://mrf.fi/dmdocuments/EVR-MRM-003.pdf.

[7] Micro Research Finland. MRF. http://mrf.fi.

[8] Marty Kraimer. MSI tool. http://www.aps.anl.gov/epics/extensions/msi/.

[9] Eric Björklund Michael Davidsaver, Jayesh Shah. mrfioc2. http://epics.sourceforge.net/mrfioc2/.

[10] PSI. driver.makefile. https://controls.web.psi.ch/cgi-bin/twiki/view/Main/DriverMakefile.

[11] PSI. mrfioc2 repository. https://github.psi.ch/projects/ED/repos/mrfioc2/browse.