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. Each EVR is 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 [14] is an EPICS device support for the Micro Research Finland (MRF) [10] 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 [3] with additional kernel modules (eg. PCIe) for communication with the hardware. This project is a continued development from the original mrfioc2 driver [12].

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.

The mrfioc2 contains all the template files needed to use the EVR in your IOC application[1]. The templates are included in the substitution file, which is in turn loaded by the IOC application (as it is common with any IOC application). It is recommended to use one of the existing EVR substitution files, available in `mrfioc2/PSI/example/*.subs`. If these are not suitable, user can adopt them to their needs. An EVR substitution file consists of

- an EVR form factor specific template. This template contains all the records describing EVR components (eg. inputs, outputs, pulsers, ...) available on a specific EVR form factor. It accepts macros that allow application specific configuration of these components. For example, through setting a correct macro, one can enable or disable selected output. The following is the EVR substitution file snippet using EVR-VME-300 form factor template. It shows how to disable front panel output 1 by substituting macro `FrontUnivOut1-Ena-SP` with 0.

```
file "$(mrfioc2_TEMPLATES=db)/evr-vme-300.db"
{
  {
    .....
    FrontUnivOut1-Ena-SP=0,
    .....
  }
}
```

- other templates, such as `evr-pulserMap.template, evr-softEvent.template, evr-specialFunctionMap.template, evr-delayModule.template`. These are used to define application specific behavior, such as mapping of events (described in Section 2.5) or defining additional components plugged in to the EVR (eg. universal I/O modules).

Each of the example EVR substitution files in `mrfioc2/PSI/example` folder contains all the available macros for the form factor it represents, together

[1] mrfioc2 files and folders overview is available in Section 3, the template and substitution files are described in Section 3.6 .

with short documentation. Besides configuring the EVR by setting the appropriate macros in the EVR substitution file, it is also possible to configure the EVR through the GUI at runtime (starting the GUI is described in Section 7).

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 the EVR 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 [8].

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 [2]

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

2.2 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.

[2] For EVR form factors that support GTX outputs (more about these is available in [8]), there is an additional macro. **ExtInhib-Sel**=*0* is used to set whether to honor the hardware inhibit signal(`0`) or not (`1`).

Substitution file macros

- **Link-Clk-SP**=*142.8* : 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`.

2.3 Pulse Generator - Pulser (Pul)

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

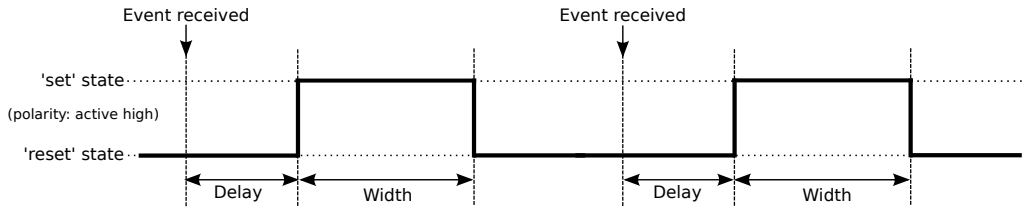

Figure 1: An example of a pulse generated after the reception of the event.

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*.

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 There are a maximum of 24 pulsers available, depending on the EVR form factor. Pulsers are named Pul#, where # is the ID of the pulser ranging, from 0 to 23.

- **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 us. The value range depends on pulse prescaler and event clock settings. Assuming event clock set to 142.8 MHz, the range is:
  - `0 us - 1971083204567.857 us`, when prescaler is set to 65535 (maximum value).
  - `0 us - 30076801.779 us`, when prescaler is set to 1 (or if the pulser does not have a prescaler).
- **Pul#-Width-SP**=*0* : Sets the pulse **width** in us. The value range depends on pulse prescaler, event clock settings and EVR form factor. Assuming event clock set to 142.8 MHz, the range is:
  - `0 us - 1971083204567.857 us`, when prescaler is set to 65535 (maximum value).
  - `0 us - 30076801.779 us`, when prescaler is set to 1.
  - `0 us - 458.929 us` for pulsers without prescalers.

- **Pul#-Prescaler-SP**=*1* : Decreases the resolution and increases range of both **delay** and **width** by an integer factor. Value range: `0-65535`. Maximum configurable prescale value depends on the EVR form factor. Pulsers without prescalers use a fixed prescale value of 1.

EVR-VME-300 also supports the following macros:

- **Pul#-Gate-Mask-SP**=*0*: Set the gate (gates are described in Section 2.3.1) used to mask this pulser. While the selected gate is activated, this pulser's **Trig** function is inhibited. Gates are selected using a bit mask, eg. value 0x1 corresponds to gate 0, value 0x2 corresponds to gate 1, value 0x3 corresponds to gate 0 and 1. Values are in range of `0x0 - 0xF`.

- **Pul#-Gate-Enable-SP**=*0*: Set the gate (gates are described in Section 2.3.1) used to enable this pulser. While the selected gate is not activated, this pulser's **Trig** function is inhibited. Gates are selected using a bit mask, eg. value 0x1 corresponds to gate 0, value 0x2 corresponds to gate 1, value 0x3 corresponds to gate 0 and 1. Values are in range of `0x0 - 0xF`.

Substitution file macros for mapping events to pulser function are available in Section 2.5.1, together with an **example**.

2.3.1 Pulser gating (gates)

Pulser gating was introduced in series 300 of the event receivers. In EVR-VME-300 form factor, there exist 4 pulsers configured as gates (pulser 28 - pulser 31), which correspond to gate 0 - gate 3. They differ from normal pulser:

- can only be enabled or disabled (**Pul#-Ena-Sel**), where setting other properties has no effect.

- only uses **Set** and **Reset** functions, not the **Trig** function of the pulser.

2.4 Outputs

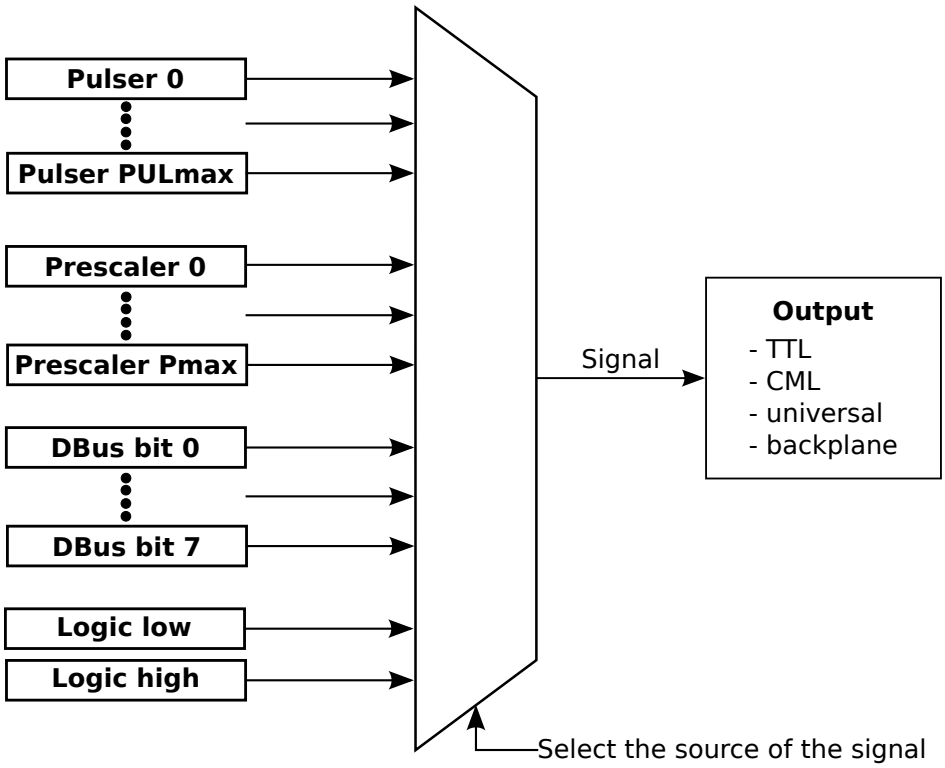

Figure 2: Available source signals for each output

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 *FrontUnivOut#*, *FrontUnivOut#* or *RearUniv#*, where the range of the number # depends on the EVR form factor. Macros for the *FrontUnivOut0* are described below.

- **FrontUnivOut0-Ena-SP**=*1* : When set to `enabled(1)` the mapping defined in `FrontUnivOut0-Src-SP` is used. When `disabled(0)`, a mapping of `Force Low(63)` from Table 1 is used.

- **FrontUnivOut0-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.

300 series of event receivers (EVR-VME-300, EVR-PCIe-300DC) also support the following macro:

- **FrontUnivOut0-Src2-SP**=*63* : Two sources for one output can be selected. The resulting output signal is a logical OR of **FrontUnivOut0-Src-SP** and **FrontUnivOut0-Src2-SP** source signals. As with **FrontUnivOut0-Src-SP**, `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 second output source signal.

2.4.1 Front Panel TTL Output (FrontUnivOut)

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

2.4.2 Front Panel CML Output (FrontUnivOut)

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 `NBIT` times the event clock rate, thus the outputs allow for producing fine grained adjustable pulses and clock frequencies. Value of `NBIT` depends on the event receiver form factor (`NBIT` is 20 in EVR-VME-230 and 40 in EVR-VME-300).

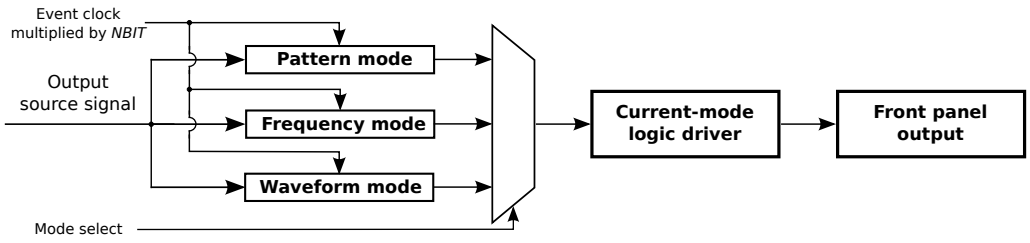

Figure 4: Overview of the front panel CML outputs

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 **pattern mode** a user configurable `NBIT`-bit pattern is 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 `NBIT` 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 $65536/NBIT$ event clock periods. Assuming event clock at 142.8 MHz and `NBIT=40`, the range is 3.676 ns - 11473.126 ns.
  - **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 $65536/NBIT$ event clock periods. Assuming event clock at 142.8 MHz and `NBIT=40`, the range is 3.676 ns - 11473.126 ns.
  - **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 $65536/NBIT$ event clock periods. Assuming event clock at 142.8 MHz, the range is 3.676 ns - 11473.126 ns.

- In **waveform mode**, one can generate arbitrary bit patterns(waveforms), triggered by the output source signal(pulser, DBus signal,...). The waveform length is in multiple of `NBIT` bits (each bit is 1/`NBIT` of the event clock period), where maximum waveform length is `NBIT`x2048 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 is an additional delay calculator 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)` and apply the waveform, 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, which yields 0 ns - 14334 ns at 142.8 MHz event clock. The sum of `delay` and `width` must not exceed this maximum value.
- **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, which yields 0 ns - 14334 ns at 142.8 MHz event clock. The sum of `delay` and `width` must not exceed this maximum value.

2.4.3 Front Panel Universal I/O (FrontUnivOut)

Universal I/O slots provide different types of input or output with exchangeable Universal I/O modules [9]. Each module provides two inputs or outputs (TTL, NIM or optical). The exact output signal is dependent on the inserted module. Figure 5, shows an example of a universal I/O module with two outputs inserted in front panel universal slot.

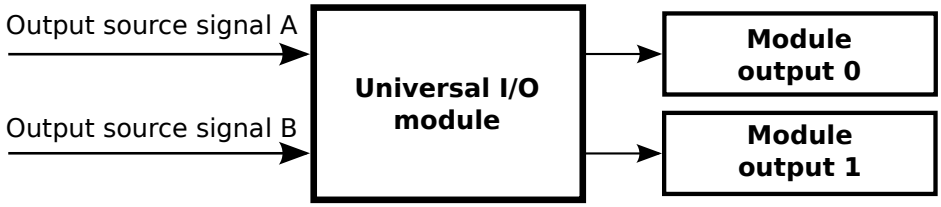

Figure 5: An example universal I/O module with two outputs.

2.4.4 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.

2.5 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.

2.5.1 Pulser mapping

Event receivers have multiple pulsers that can preform several functions upon reception of events (described in Section 2.3). Each pulser-function combination can be mapped to multiple events.

Substitution file macros:

- **SYS**=*MTEST-VME-EVRTEST* : The system name.
- **DEVICE**=*EVR0* : The name of the connected Event Receiver / timing card, 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 2.3.
- **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,                  DEVICE,  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 }
}
```

2.5.2 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.
- **DEVICE**=*EVR0* : The name of the connected Event Receiver / timing card, 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).
- **FLNK** if specified, forward links to the specified record after all records from this template are processed.

Example 1: 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,                   DEVICE,   EVT,    CODE}
        {"MTEST-VME-EVRTEST",   "EVR0",    "1",    "1"}
        {"MTEST-VME-EVRTEST",   "EVR0",    "2",    "2"}
}
```

Example 2: The same as example 1, except that a forward link is used to process record with name `LINKED_RECORD`, after event 2 from the timing system is received.

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

Another template exists, that enables us to measure performance of the received events in addition to mapping to EPICS events and forward linking to other records. The `evr-softEvent-measure.template` is an in-place replacement of the `evr-softEvent.template`. It is important to note that both templates **must not** be used for the same event from the timing system (`$(EVT)` macro)!

The template exposes the following records:

- `$(SYS)-$(EVR):Event-$(EVT)-PERF-MAX-I` stores the maximum time in [ms] between two `$(EVT)` event arrivals. It must be manually initialized at the start of the measurement.
- `$(SYS)-$(EVR):Event-$(EVT)-PERF-MIN-I` stores the minimum time in [ms] between two `$(EVT)` event arrivals. It must be manually initialized at the start of the measurement.
- `$(SYS)-$(EVR):Event-$(EVT)-HIST-I` creates a histogram of times between two `$(EVT)` event arrivals in [ms]. By default it uses 200 bins with high limit of 12 ms and low limit of 8 ms.

Substitution file macros in addition to macros used in `evr-softEvent.template`:

- **HIST_LEN**=*200* : Number of bins for the histogram. Default: 200 ms
- **HIST_ULIM**=*12* : Histogram upper limit in [ms]. Default: 12 ms
- **HIST_LLIM**=*8*: Histogram lower limit in [ms]. Default: 8 ms

NOTE: make sure your system supports the 'histogram' record before using this template! On PSI system this is achieved by adding `require histogram` to your startup script, before loading this template.

Example 3: The same as example 1, except that performance measurements are preformed for events 1 and 2. Histogram record is set to use 200 bins that store times between two event receptions in the range of 8 ms to 12 ms.

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

2.5.3 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 2.13 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 2.12 for details. (default event 125)
- **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 2.12 for details. (default event 124)

- **Shift 0** : Shifts the current value of the shift register up by one bit and sets the low bit to 0. See Section 2.12 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 2.12 for details. (default event 113)
- **FIFO** : Bypass the automatic allocation mechanism and always include this code in the FIFO memory. See Section 2.12 for details.

Substitution file macros:

- **SYS**=*MTEST-VME-EVRTEST* : The system name.
- **DEVICE**=*EVR0* : The name of the connected Event Receiver / timing card, 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,                    DEVICE,   EVT,   FUNC }
        {"MTEST-VME-EVRTEST",   "EVR0",    "1",   "Blink"}
        {"MTEST-VME-EVRTEST",   "EVR0",    "6",   "Blink"}
}
```

2.6 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 8 prescalers available, depending on the EVR form factor. Prescalers are named PS#, where # is the ID of the prescaler, ranging from 0 to 7.

- **PS#-Div-SP**=*2* : Sets the integer divisor between the event clock and the prescaled event clock output in a range of `2-4294967295`. Maximum configurable prescale value depends on the EVR form factor.

EVR-VME-300 also supports the following macros:

- **PS#-PulserMap-L-SP**=*0* : Trigger a pulser on prescaler rising edge. Pulsers 0-15 are bit-wise selectable. Eg. value 0x1 triggers pulser 0, 0x2 triggers pulser 1, 0x3 triggers pulsers 0 and 1.
- **PS#-PulserMap-H-SP**=*0* : Trigger a pulser on prescaler rising edge. Pulsers 16-23 are bit-wise selectable. Eg. value 0x1 triggers pulser 16, 0x2 triggers pulser 17, 0x3 triggers pulsers 16 and 17.

2.7 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(FrontUnivOut, RearUniv, ...), or used as inputs (described in Section 2.8).

2.8 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`.

2.9 Universal and rear (trnasition board) inputs

Based on the EVR form factor, universal input modules can be inserted into EVR universal I/O slots, on the transition board (eg. VME timing card) or on a special extension module (eg. PCIe form factor timing card). These inputs have the same settings as front panel TTL inputs (see Section 2.8), except that they are not named with prefix `FPIn`, but rather with:

- `UnivIn` in case of universal inputs and
- `RearIn` in case of rear inputs.

2.10 UNIV-LVPECL-DLY high precision delay module

EVR has universal I/O slots, described in Section 2.4.3, 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 2.4, 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.
- **DEVICE**=*EVR0* : The name of the Event Receiver / timing card, 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,
      DEVICE  = EVR0,
      SLOT    = 0,

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

      Enabled = 0,
      Delay0  = 5,
```

```
      Delay1  = 5,
    }
}
```

2.11 Delay compensation

Series 300 of event receivers (eg. EVR-VME-300, EVR-PCIe-300) are able to receive and process delay compensation beacons from the event generator. They are used to measure the time it takes for the signal to arrive from the event generator to the event receiver (`path delay`). After measuring the `path delays` across the entire timing network, user should set the `target delay` on each event receiver to the value that is slightly higher than the maximum `path delay`. This way each event receiver will automatically compensate for the signal delay caused by various cable lengths and temperature variations across the timing network in order to ensure that all events and data across the timing network are processed at the same time.

Substitution file macros

- **DlyCompensation-Enabled-Sel**=*1* : `Enable (1)` or `disable (0)` the delay compensation.
- **DlyCompensation-Target-SP**=*1000* : Target delay compensation value in [ns].
- **DlyCompensation-Source-I**=*"SIN-TIMAST-TMA:EvrDC-SP CP"* : Points to a record which will be used to set target delay compensation value for this EVR. If set to empty, remote setting of delay compensation target value is not used. By the default the value points to a record used in PSI SwissFEL facility.
- **DlyCompensation-Source-Disa**=*0* : `Enables (0)` or `disables (1)` sourcing the delay compensation target value from record specified with 'DlyCompensation-Source-I' macro

2.11.1 Getting target delay compensation from external record

When there are many event receivers in the timing network, it is convenient if target delay compensation can be set for all the event receivers to the same value at the same time. In order to use this functionality, let `DlyCompensation-Source-I` macro in the substitution file point to the record which will be used to set target delay compensation value for EVRs

in the timing network. Assuming this record is `SIN-TIMAST-TMA:EvrDC-SP`, the substitution file macro looks like this:

```
DlyCompensation-Source-I="SIN-TIMAST-TMA:EvrDC-SP CP"
```

Target delay compensation of this EVR will be automatically updated to `SIN-TIMAST-TMA:EvrDC-SP` value, whenever it changes (because we have CP link). In order to disable automatic update simply set `DISA` field of `$(SYS)-$(DEVICE):DlyCompensation-Source-I` record to `1`. The same setting is also exposed as a macro in the substitution file. To disable automatic update of the target delay compensation value, set the following macro in the substitution file to `1`:

```
DlyCompensation-Source-Disa=0
```

2.12 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 2.5.3). 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 2.5.3.
  - `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`.

2.13 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.

3 mrfioc2 organization

3.1 Documentation

- `documentation/` 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. Mind that the tutorials are tailored to PSI users, though the functionality and settings of the event receiver are general.

3.2 Source code

- `evgMrmApp/src` contains source files for the an Event Generator.
- `evrMrmApp/src` contains source files for an Event Receiver.
- `mrfCommon` contains common functions and definitions used in the mrfioc2 device support.
- `mrfShared` contains code shared between EVG and EVR together with some OS specifics and kernel modules. Data buffer handling is also implemented here.
- `/` contain makefiles for building the mrfioc2 device support.
- `configure` and `iocBoot` are standard IOC application folders [5].
- `mrfApp` folder contains source and makefiles that builds an executable IOC for this project, when using standard EPICS build system [5].

3.3 Templates and substitution files

- `evrMrmApp/Db/evr-vme-230.substitutions` is an EVR form factor specific substitution file used to generate the form factor specific template file for EVR-VME-230. It is automatically generated when building the project.
- `evrMrmApp/Db/evr-vme-300.substitutions` is an EVR form factor specific substitution file used to generate the form factor specific template file for EVR-VME-300. It is automatically generated when building the project.

- `evrMrmApp/Db/evr-embedded.substitutions` is an EVR form factor specific substitution file used to generate the form factor specific template file for embedded event reciver in event master (EVG-VME-300). It is automatically generated when building the project.

- `evrMrmApp/Db/evr-pcie-300.substitutions` is an EVR form factor specific substitution file used to generate the form factor specific template file for EVR-PCIe-300. It is automatically generated when building the project.

- `evrMrmApp/Db/evr-pcie-300DC.substitutions` is an EVR form factor specific substitution file used to generate the form factor specific template file for EVR-PCIe-300DC (PCIe version of EVR with delay compensation). It is automatically generated when building the project.

- `evrMrmApp/Db/evr-cpci-230.substitutions` is an EVR form factor specific substitution file used to generate the form factor specific template file for EVR-cPCI-230. It is automatically generated when building the project.

3.4 GUI

- `evrMrmApp/opi/EVR/start_EVR.sh` is a script used to launch the caQtDM [1] EVR GUI.

- `evrMrmApp/opi/EVR/start_EVR-health.sh` is a script used to launch the caQtDM [1] EVR health monitor GUI.

- `evrMrmApp/opi/EVR` folder contains caQtDM [1] GUI.

- `evrMrmApp/opi/medm` folder contains deprecated medm [7] GUIs (they do not work due to many database changes).

3.5 PSI specific example folder

- `PSI/example` contains example startup scripts and example substitution files that can be used in creation of an IOC application (for each form factor separately). Example substitution files also contain all the form factor specific macros available to set up the EVR or EVG. There is one additional substitution file `evg_VME-300-fout.subs` tailored for EVG-VME-300 series that operate as fanouts/concentrators. It differs from the default EVG-VME-300 series substitution file (`evg_VME-300.subs`)

only in event clock source selection setting (`EvtClk-Source-Sel` macro), which is set to `Recovered RX clock`.

3.6 Macros, substitution files and template files

The mrfioc2 device support contains a series of template and substitution files for the EVR, located in `mrfioc2/evrMrmAp/Db` and `mrfioc2/mrmShared/Db` folder. To simplify deployment of an IOC application, form factor specific templates are automatically generated uppon build time 4. They can be directly included in the EVR substitution file and allow the usage of macros to set application specific configuration of the EVR. User can define its own EVR substitution file, or use one of the examples in `mrfioc2/PSI/example` folder (evr_VME-300.subs, evr_VME-230.subs, evr_PCIe-300.subs ...).

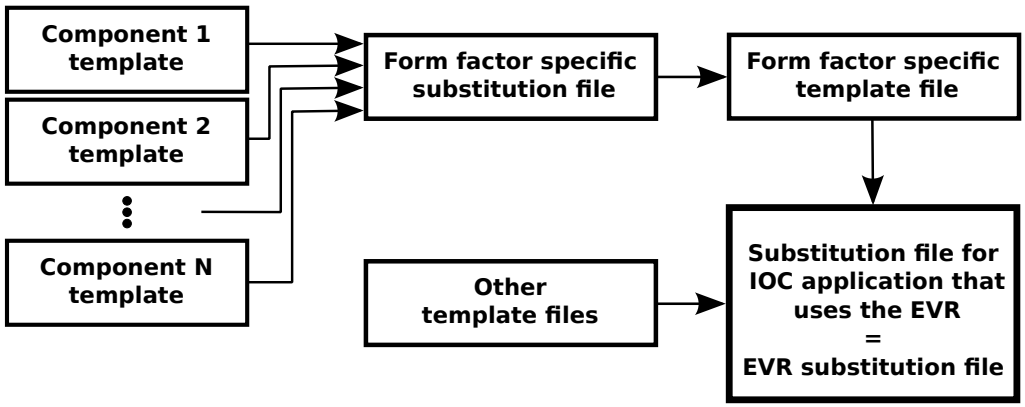

Figure 6: Connection between template and substitution files for the EVR.

Figure 6 shows how the templates and substitutions are connected.

- *Component templates* [3] contain records corresponding to specific EVR component (eg. inputs, outputs,...). Each EVR form factor supports different components and a different number of each component (exact number of components is available in [8]).

- Depending on the EVR form factor, appropriate component templates are listed in a *form factor specific substitution file.* [4].

[3] evr-base.template, evr-cml.template, evr-in.template, evr-out.template, evr-prescaler.template, evr-pulser.template, sfp.template, ...

[4] evr-vme-300.substitutions, evr-pcie-300.substitutions, ...

- *Form factor specific template files* [5] are are automatically generated uppon build time 4. These contain all the records for an EVR form factor, as listed in the form factor specific substitution file (eg. records for 2 inputs, 3 prescalers, ...). This template should be included in the EVR substitution file for your IOC application, since it only contains form factor specifics. The application specific behavior can be set using macros. The template accepts macros that define system name, EVR name and macros that define some of the record field values for each component. For example, through setting a correct record field value for a desired output, one can enable or disable this output. Such a template can also be created manually using the MSI tool from a form factor specific substitution file, as described in Section 5.

- *Other templates* [6] define application specific behavior and should be included in the EVR substitution file for your IOC application.

[5] evr-vme-230.db, evr-pcie-300.db, ...

[6] evr-pulserMap.template, evr-softEvent.template, evr-specialFunctionMap.template, evr-delayModule.template, ...

4 Building from scratch

There are two possible ways to build the mrfioc2 device support. First is using the driver.makefile [13] on the PSI infrastructure, and second using the standard makefile [5].

Prerequisites

- EPICS Base 3.14.x [6].
- devLib2 2.6 [3].
- git [2], to clone the repository [14].
- MSI tool [11] for expanding databases.

4.1 PSI driver.makefile

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

```
git clone git@git.psi.ch:epics_drivers/mrfioc2.git
```

and then:

1. Move to the folder, which contains mrfioc2 sources:

   ```
   cd mrfioc2
   ```

2. Switch to the latest version, which is `3.0.2` at the time of writing this manual:

   ```
   git fetch
   git checkout 3.0.2
   ```

3. compile the device support:

   ```
   make
   ```

4. generate templates:

   ```
   make db
   ```

5. install the compiled module (mrfioc2 device support and template):

```
make install
```

Compilation will only work on a PSI infrastructure where the correct applications and environment variables are already configured. MSI tool [11] is required in order to generate templates. Installed templates should be used for your IOC application, though they can be created separately, following the instructions in Section 5.

Installed module can then be used with one of the example startup scripts 3.5.

4.2 Standard makefile

To compile the mrfioc2 as a standard IOC application, it is suggested to first clone the latest mrfioc2 repository from the PSI git:

```
git clone git@git.psi.ch:epics_drivers/mrfioc2.git
```

Then follow these steps:

1. Move to the folder, which contains mrfioc2 sources:

   ```
   cd mrfioc2
   ```

2. Switch to the latest version, which is `3.0.2` at the time of writing this manual:

   ```
   git fetch
   git checkout 3.0.2
   ```

3. Adjust the `mrfioc2/configure/RELEASE` file for your environment. Make sure that `DEVLIB2` points to your devLib2 and `EPICS_BASE` points to your EPICS base.

4. Run `make -f Makefile` in `mrfioc2` folder.

   ```
   make -f Makefile
   ```

5. Template files in `mrfioc2/Db` folder should be used for your IOC application, though they can be created separately, following the instructions in Section 5.

As mrfioc2 compiles as a standard EPICS application, user can inspect `iocBoot/iocMrf/` folder for startup script examples.

5 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 generated during the build process.

Whenever possible use the automatic template generation that occurs during the build process!

To create such a template, use a form factor specific substitution file(eg. file list items in 3.3) and expand it using the MSI tool [11]. The tool works the same way as the EPICS macro substitutions do. The output of the tool can then be used in the substitution file for your IOC application.

However, in order to set the default values of macros, while recursively expanding templates (using MSI), a trick is used in the original template files.

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

Solution The record fields in template files are encapsulated using nested macro, like so:
`field( VAL , "$($(OBJ)$(ID)-Div-SP\=2)")` Here is what happens:

- `$(OBJ)$(ID)` 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 `OBJ=PS, ID=0` yields
  `field( VAL , "$(PS0-Div-SP=2,undefined)")`
- The macro substitution `PS0-Div-SP=value` can then be used in our application substitution file to set the field `value`, otherwise it defaults to '2'.

Individual form factor compatible template files can also be generated. The following **example** shows how to create a VME-EVR-300 form factor compatible template file, by issuing the following command in `mrfioc2` folder:

```
msi -I evrMrmApp/Db/ -I mrmShared/Db/
    -S evrMrmApp/Db/evr-vme-300.substitutions
    -o evrMrmApp/Db/evr-vme-300.db
```

The command will use the EVR form factor specific `evr-vme-300.substitutions` file to generate the template `evr-vme-300.db` to be used in your IOC application substitution file.

6 Firmware update

Take great care when flashing the firmware on your device. Wrong bit-files or errors while flashing will most likely render your device unusable!

mrfioc2 EPICS device support enables us to update the VME, PCIe and cPCI event generator or event receiver firmware remotely. There are two ways to update the firmware, either through iocsh functions (recommended way) or through EPICS records.

6.1 Usage of the iocsh functions for accessing the device flash memory

In order to read back the content of the flash chip on the device, use the following iocsh command:

```
mrmFlashRead Device File
```

where:

- `Device` is the device name we wish to read the flash chip memory from. Eg.: EVR0, EVR1, EVG0, EVG1
- `File` is the path to the bit-file where the content of the flash chip memory will be written to.

In order to flash firmware to the device, use the following iocsh command:

```
mrmFlashWrite Device File
```

where:

- `Device` is the device name we wish to flash. Eg.: EVR0, EVR1, EVG0, EVG1
- `File` is the path to the bit-file that contains the firmware we wish to write to the flash chip on the device.

Flash chip information and access status can be printed using the following iocsh command:

```
mrmFlashStatus Device
```

where:

- `Device` is the device name we wish to print the information for. Eg.: EVR0, EVR1, EVG0, EVG1

Example: Reading the information of the EVR0 device flash chip by issuing `mrmFlashStatus EVR0` command can produce the following output:

```
Status report of EVR0 flash chip
    Memory size: 16777216 bytes = 16384 kB = 16 MB
    Sector size: 262144 bytes = 256 kB
    Page size: 256 bytes
    First address for reading / writing is at offset: 0 [bytes]
    Flash is currently not being accessed.
    Last completed flash operation was not successful.
    Last completed read operation was not successful.
```

The output contains :

- basic information about the flash chip (memory, sector and page size),
- read / write offset. This is the offset in the flash chip memory, where reading and flashing starts.
- Indication whether reading or flashing is currently in progress (either through EPICS records or through iocsh functions).
- Indication whether the last completed reading and flashing commands were successful (commands can be issued through EPICS records or through iocsh functions). If no read or write operations were issued yet, the indication states `not successful`.

In addition to the functions described above, an iocsh variable `mrfioc2_flashDebug` is used to set the verbosity level of debug messages that are printed to the iocsh when using flashing functions. Setting the variable to value `0` shows the least information, where setting it to `255` shows the most information.

Example: Set debug messages verbosity to 3 from the iocsh:

```
var mrfioc2_flashDebug 3
```

6.2 Using EPICS records for accessing the flash memory

A template file `flash.template` contains records that expose flash memory access. It be loaded by adding the following lines to the EVR (or EVG) substitution file:

```
## Flash access support
## Uncomment this substitution to load records that expose
## read and write access to the flash chip on the device.
#
# Macros:
#  SYS = System name (auto expanded by parent)
#  DEVICE = Event receiver / timing card name (same as mrmEvrSetupVME())
#           Eg. EVR0. (auto expanded by parent)
#
file "$(mrfioc2_TEMPLATES=db)/flash.template"
{
{}
}
```

When using example substitution file provided for the PSI infrastructure the snippet above is already included, but commented out. In order to use it, uncomment it. The following records are then available for status reporting:

- `$(SYS)-$(DEVICE):Flash-IsOffsetValid-RB` record. Indicates if the offset was set correctly. The offset determines the address on the flash chip where reading/writing starts from/to. It is set automatically based on the device form factor. It can also be set manually through an iocsh command, though this is dangerous. See Section 6.3 for details.

- `$(SYS)-$(DEVICE):Flash-InProgress-RB` record. When processed, indicates if reading or flashing is currently in progress (either through records or through iocsh functions).

- `$(SYS)-$(DEVICE):Flash-Flash-RB` record. When processed, indicates if the last completed flashing command was successful (flashing command can be issued through EPICS records or through iocsh functions).

- `$(SYS)-$(DEVICE):Flash-Read-RB` record. When processed, indicates if the last completed read command was successful (read command can be issued through EPICS records or through iocsh functions).

The following procedure enables us to read back the content of the flash chip:

1. Check if the offset was set correctly by reading `$(SYS)-$(DEVICE):Flash-IsOffsetValid-RB` record.

2. Use `$(SYS)-$(DEVICE):Flash-Filename-SP` record to set the path to the bit-file where the content of the flash chip memory will be written to.

3. Use `$(SYS)-$(DEVICE):Flash-Filename-RB` record to check the previously set path to the bit-file.

4. Process `$(SYS)-$(DEVICE):Flash-Read-Cmd` record to initiate the reading procedure.

5. Periodically process `$(SYS)-$(DEVICE):Flash-InProgress-RB` record in order to see if the procedure has finished. Mind that flash access is slow.

6. After reading had finished, process `$(SYS)-$(DEVICE):Flash-Read-RB` to see if the operation was successful.

The following procedure enables us to flash the firmware:

1. Check if the offset was set correctly by reading `$(SYS)-$(DEVICE):Flash-IsOffsetValid-RB` record.

2. Use `$(SYS)-$(DEVICE):Flash-Filename-SP` record to set the path to the bit-file that contains the firmware you wish to write to the flash chip on the device.

3. Use `$(SYS)-$(DEVICE):Flash-Filename-RB` record to check the previously set path to the bit-file.

4. Process `$(SYS)-$(DEVICE):Flash-Flash-Cmd` record to initiate the flash procedure.

5. Periodically process `$(SYS)-$(DEVICE):Flash-InProgress-RB` record in order to see if the procedure has finished. Mind that flash access is slow.

6. After flashing had finished, process `$(SYS)-$(DEVICE):Flash-Flash-RB` to see if the operation was successful.

6.3 Advanced

Sometimes a device with an old, broken or alien firmware needs to be flashed. In these cases the driver will exit with an error, since not all needed parameters could be read from the card. There is an optional `ignore version`

`error` parameter that can be passed to `mrmEvgSetupPCI`, `mrmEvgSetupVME`, `mrmEvrSetupPCI` or `mrmEvrSetupVME` commands in order to ignore such errors while setting up the driver. The following startup script[7] example shows the usage of this parameter:

```
require mrfioc2

#########################
#-----! EVR Setup ------!#
#########################

mrmEvrSetupVME(EVR0, 3, 0x3000000, 5, 0x26, true);
```

This startup file will:

1. Load the latest version of the mrfioc2 driver
2. Setup the VME event receiver using `mrmEvrSetupVME` command:
   (a) Set device name to `EVR0`.
   (b) Set VME crate slot to `3`.
   (c) Set the base A32 address (memory offset) to `0x3000000`.
   (d) Set interrupt level to `5`.
   (e) Set interrupt vector to `0x26`.
   (f) Finally, set the optional `ignore version error` parameter to `true`.

When the `ignore version error` parameter is set (it's value is `true` or a non-zero number), the device setup procedure will only notify the user of potential version mismatches instead of exiting the driver and interrupts on PCI devices will not be enabled and connected. Not all parts of the driver are guaranteed to work with `ignore version error` enabled, but flashing functions described in Section 6.1 can still be used.

In addition to the flashing functions described in Section 6.1, there is another iocsh function that allows us to override the offset from the start of the flash chip memory where reading/writing begins in order to force flashing of the firmware to the device. In normal operation, the offset is auto detected based on the device form factor. The auto detection can fail in rare cases where device firmware is not supported by this driver. This can be the case

[7] This is an example for the PSI infrastructure, but the setup command is the same in all cases

with old, broken or alien firmwares. **This is the only case where using this function should be considered.**

> Setting wrong offset for flashing the device is very dangerous, and should be left to auto detection whenever possible!

In order to set the offset, use the following iocsh command:

```
mrmFlashSetOffset Device Offset
```

where:

- `Device` is the device name we wish to flash. Eg.: EVR0, EVR1, EVG0, EVG1
- `Offset` is the offset from the beginning of the flash chip memory to write to / read from.

7 GUI

There is a caQtDM [1] GUI for the Event Generator and Event Receiver available in `mrfioc2/evgMrmApp/opi/EVG` and `mrfioc2/evrMrmApp/opi/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:

```
-d <EVR name> ......... set the event receiver / timing card name
                        (default:EVR0)

-f <form factor> ...... choose the event receiver form factor
                        (default: VME-300)
                        Choices: VME, PCIe, VME-300, PCIe-300DC

-n ..................... Do not attach to existing caQtDM.
                        Open new one instead

-h    .................. shows the options and usage
```

Example: Open the GUI for the EVR-VME-300 event receiver named `EVR0`, using system name `MTEST-VME-EVRTEST`.

```
cd mrfioc2/evrMrmApp/opi/EVR
./start_EVR.sh -s MTEST-VME-EVRTEST
```

References

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

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

[3] devLib extensions for PCI and VME64x bus access. https://github.com/epics-modules/devlib2.

[4] Tutorial. https://git.psi.ch/epics_driver_modules/mrfioc2/raw/n/documentation/tutorial.pdf.

[5] EPICS Application Development Guide. http://www.aps.anl.gov/epics/base/R3-14/12-docs/AppDevGuide/.

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

[7] Kenneth Evans. MEDM. http://www.aps.anl.gov/epics/extensions/medm/.

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

[9] Micro Research Finland. Universal I/O modules. http://mrf.fi/index.php/universal-io-modules.

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

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

[12] Eric Björklund Michael Davidsaver, Jayesh Shah. mrfioc2. https://github.com/epics-modules/mrfioc2/.

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

[14] Jure Krašna Michael Davidsaver Jayesh Shah Eric Björklund Sašo Skube, Tom Slejko. mrfioc2 repository. https://git.psi.ch/epics_driver_modules/mrfioc2/tree/on.

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
16	Pulser 16
17	Pulser 17
18	Pulser 18
19	Pulser 19
20	Pulser 20
21	Pulser 21
22	Pulser 22
23	Pulser 23
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