Introduction to the Timing System

April 13, 2015

Abstract

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, generating pulses, synchronized clock or custom signals on its outputs. This document contains description of the Event Receiver components and tutorials containing step-by-step instructions to configuring some of the basic functionalities of the Event Receiver

Contents

1 Event Receiver **2**
1.1 Events and event clock 2
1.2 Distributed Bus (DBus) 2
1.3 Pulse Generator - Pulser (Pul) 2
1.4 Prescaler (PS) 3
1.5 Front Panel TTL Input (FPIn) 3
1.6 Outputs 4
1.6.1 Front Panel TTL Output (FrontOut) 4
1.6.2 Front Panel CML Output (FrontOut) 4
1.6.3 Front Panel Universal I/O (FrontUnivOut) 5
1.6.4 Rear Universal output - Transition Board output (RearUniv) 5

2 EVR Tutorials **5**
2.1 Create a new IOC application 6
2.2 Generate a pulse uppon receiving an event 7
2.2.1 Instructions: 7
2.2.2 Substitution snippet explanation: 8
2.3 Process an EPICS record uppon receiving an event 8
2.3.1 Instructions: 8
2.3.2 Substitution snippet explanation: 9
2.4 Generate a clock signal 9
2.4.1 Instructions: 9
2.4.2 Substitution snippet explanation: 10
2.5 Output a Distributed Bus bit 10
2.5.1 Instructions: 10
2.5.2 Substitution snippet explanation: 11

1 Event Receiver

EVRs are available in various form factors, each supporting the same basic functionality, but with different number of input, output and internal components. Each of theese components is configurable by the user and is briefly described in the following chapters. There are two methods for configuring EVR components:

- setting values of macros in a substitution file (tutorials will focus on this method), or
- configuration through GUI during runtime.

The VME-EVR-230RF form factor has the following component support:

- 16 Pulse Generators, named Pul0 - Pul15. Pul0 - Pul3 have additional prescalers.
- 3 Prescalers, named PS0 - PS2
- 2 Front Panel TTL inputs, named FPIn0 - FPIn1
- 4 Front Panel TTL Outputs, named FrontOut0 - FrontOut3
- 3 Front Panel CML Outputs, named FrontOut4 - FrontOut6
- 4 Front Panel Universal I/O in two slots. FrontUnivOut0 and FrontUnivOut1 belong to slot 0, where FrontUnivOut2 and FrontUnivOut3 belong to slot 1. Each slot has 4 additional GPIO pins.
- 16 Rear universal outputs, named RearUniv0 - RearUniv15

1.1 Events and event clock

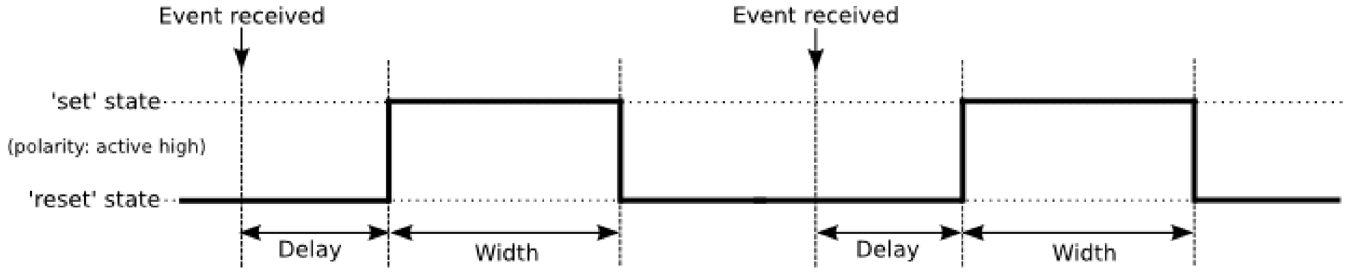
EVR receives events from an optical stream transmitted by an Event Generator. Event clock is the frequency, at which the events are received. All the EVRs lock to the phase and frequency of the EVG, thus the event clock is synchronized across the entire timing system.

1.2 Distributed Bus (DBus)

The distributed bus is able to carry 8 signals, that are proppagated throughout the timing network with the event clock rate. Individual DBus signals (bits) can be outputted through programmable outputs(FrontOut, FrontUnivOut, RearUniv).

1.3 Pulse Generator - Pulser (Pul)

A Pulse generator can be configured to output a pulse, by switching between set and reset states. The states determine the logic level of the pulser output signal.



It 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.

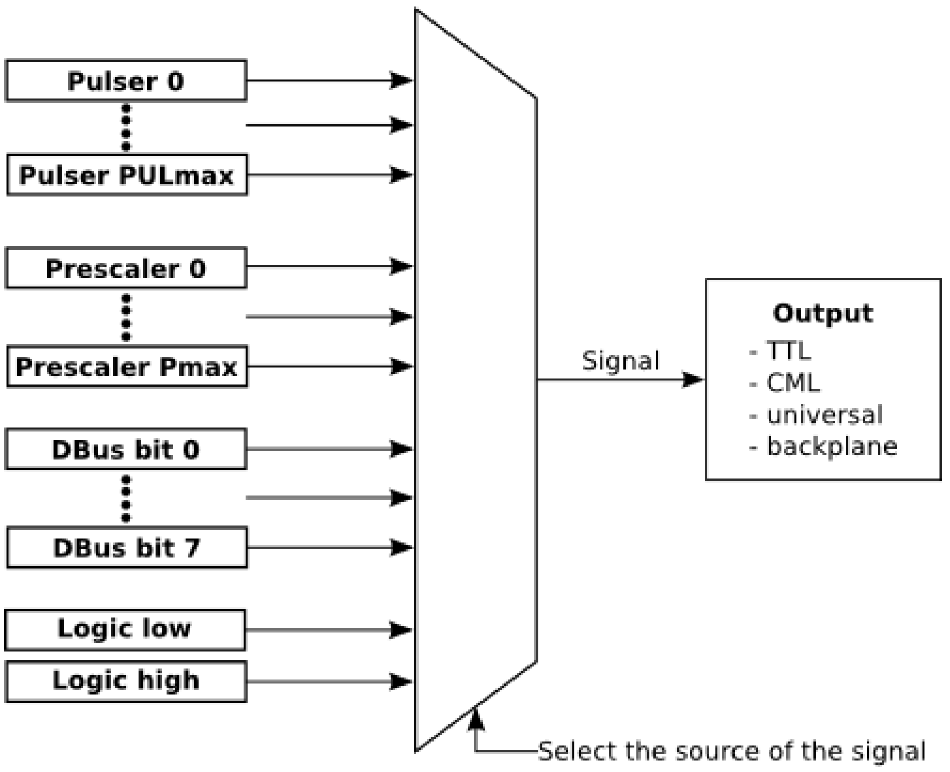
1.4 Prescaler (PS)

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.

1.5 Front Panel TTL Input (FPIn)

Front panel inputs can also be called External Event Inputs, because we can configure them to cause an event. The event can be local to the EVR or sent through the timing network. These events are handled as any other received events. Front panel inputs also provide configuration options for DBus signal manipulation.

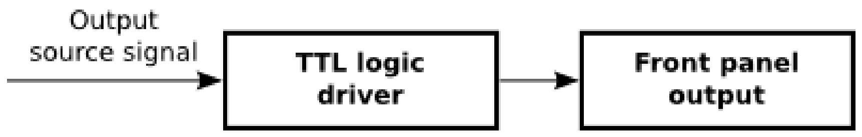
1.6 Outputs



An EVR has a number of outputs, each with a configurable source signal. 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 outputted directly, further manipulated or used as a trigger for a special function of that output.

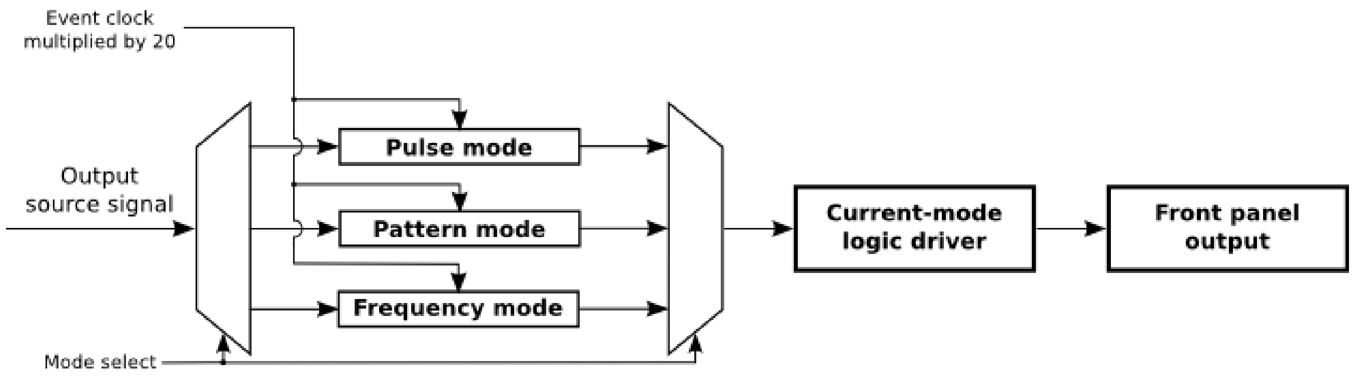
1.6.1 Front Panel TTL Output (FrontOut)

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 outputted.



1.6.2 Front Panel CML Output (FrontOut)

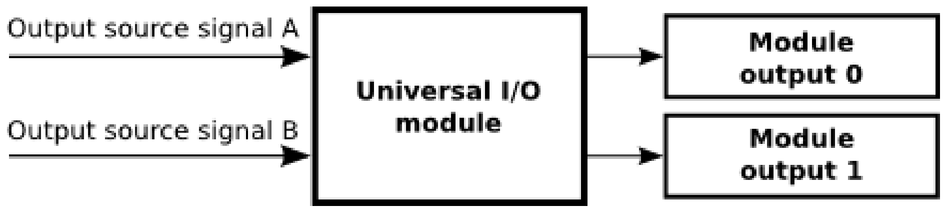
These outputs are capable of driving Current-mode logic compatible signals.



Based on the output source signal, various patterns can be outputted. A pattern is sent out at 20 times the bit rate of the event clock, thus the outputs allow for producing fine grained adjustable pulses and clock frequencies. Patterns are defined by one of the configurable CML modes:

- In **pulse mode**, the output logic monitors, and distinguishes between the states of the output source signal(logic low, rising edge, logic high, falling edge). Based on this state, a user configurable 20-bit pattern is sent out.
- In **frequency mode**, clocks with clock period in steps of 1/20 of the event clock period can be generated. The clock output is synchronized by the output source signal(pulser, DBus signal, ...)
- In **pattern mode**, one can generate arbitrary bit patterns, triggered by the output source signal(pulser, DBus signal,...). The pattern length is in multiple of 20 bits (each bit is 1/20 of the event clock period), where maximum pattern length is 20x2048 bits.

1.6.3 Front Panel Universal I/O (FrontUnivOut)



Universal I/O slots provide different types of input or output with exchangeable Universal I/O modules [3]. Each module provides two inputs or outputs (TTL , NIM or optical). The exact output signal is dependent on the inserted module.

1.6.4 Rear Universal output - Transition Board output (RearUniv)

An EVR has additional rear universal outputs. They are located on the back-plane of the EVR board. The exact output signal is dependent on the universal module inserted in the transition board.

2 EVR Tutorials

The tutorial section starts with a quick hardware checklist and shows how to create a new IOC application. The chapters following, contain step-by-step guides to configuring some of the basic functionalities of the Event Receiver.

To set up a timing system we need a VME crate, a Single Board Computer (SBC) and an EVR. A VME crate has a number of slots where SBC, EVR and other components can be inserted. Slot numbering should be checked with the VME crate documentation. To use a VME crate for the turorials, check that:

- a VME64x IFC 1210 Single Board Computer is inserted into VME crate slot 1 (how to set up IFC 1210 [4])
- an EVR is inserted in slot 2
- the EVR is connected to the timing network through an optical cable.

2.1 Create a new IOC application

To set up an IOC application for Event Receiver we need a startup script and a substitution file. The following steps demonstrate how to prepare such a SWIT compatible IOC application:

1. Create a **project folder**, eg. `MTEST-VME-EVRTEST` and a **sub-folder named** *cfg* in your project folder: `MTEST-VME-EVRTEST/cfg/`

    ```
            mkdir MTEST-VME-EVRTEST
            cd MTEST-VME-EVRTEST
            mkdir cfg
    ```

2. Copy the **startup script** to your project. Use the following command in your project folder:

    ```
            cp /fin/devl/iocBoot/templates/slejko/EVR_VME.startup
        MTEST-VME-EVRTEST_startup.script
    ```

    The startup script should look similar to

    ```
    ##System configuration
    epicsEnvSet SYS MTEST-VME-EVRTEST

    ### EVR configuration
    #epicsEnvSet EVR EVR0 ##EVR name (default EVR0)
    #epicsEnvSet EVR_SLOT 2 ##EVR SLOT (default slot 2)

    < $(TEMPLATE\_DIR)/EVR_VME.startup

    ## END OF EVR configuration
    ```

    System name (**SYS**) variable must be defined. When ommited, the **EVR** name and **EVR_SLOT** will fallback to default values. Using the above startup script, the system name is set to *MTEST-VME-EVRTEST*, and the Event Receiver named *EVR0* is placed in the physical slot 2 of the VME crate. Line "< *$(TEMPLATE_DIR)/EVR_VME.startup*" includes a generic EVR startup script, so it should not be changed by the user.

3. Copy the **substitution file** [2] to `MTEST-VME-EVRTEST/cfg/EVR.subs`. This substitution file can always be used as a starting point for new applications. Use the following command in your project folder:

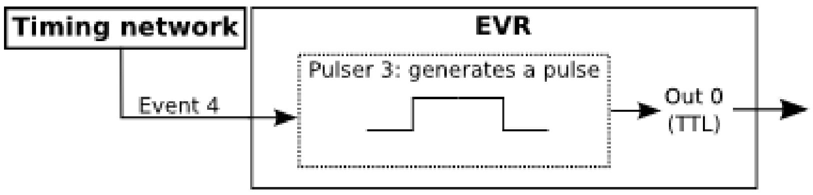
    ```
            cp /fin/devl/iocBoot/templates/slejko/EVR.subs cfg/
    ```

The macro definitions in the substitution file are used to configure the EVR. All the available macros are already present in the substitution file [2] and set to their default values, so the user can simply change the desired values, remove unused macros (same effect as leaving them with their default value). Detailed description of the substitution file is available in the EVR manual [1].

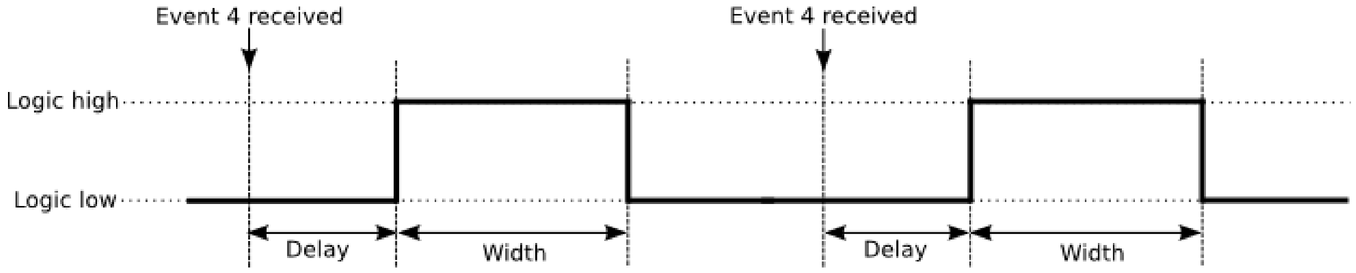
Command summary for creating a new IOC application:

```
    mkdir MTEST-VME-EVRTEST
    cd MTEST-VME-EVRTEST
    mkdir cfg
    cp /fin/devl/iocBoot/templates/slejko/EVR_VME.startup
MTEST-VME-EVRTEST_startup.script
    cp /fin/devl/iocBoot/templates/slejko/EVR.subs cfg/
```

2.2 Generate a pulse uppon receiving an event



This tutorial demonstrates how to configure an EVR to generate a 80ns wide pulse, 40ns after each occurance of event 4 and output it through the front panel TTL output 0(FrontOut0). In order to achieve this, pulser 3 delay and width are configured by setting macro value of `Pul3-Delay-SP` to 40 and of `Pul3-Width-SP` to 80. The value of the output source macro `FrontOut0-Src-SP` is set to 3, which configures the front panel output 0(FrontOut0) to use pulser 3 as its source. Finally pulser 3 is set to trigger on event 4 by setting macro value `EVT` to 4 in *evr-pulserMap.template*.



2.2.1 Instructions:

1. Create a **project folder**, eg. `MTEST-VME-EVRTEST`.

2. Copy **startup script** and an example **substitution file** to your project. Use the following commands in your project folder:

```
    cp /fin/devl/iocBoot/templates/slejko/EVR_VME.startup
MTEST-VME-EVRTEST_startup.script
    cp /fin/devl/iocBoot/templates/slejko/EVR.subs cfg/
```

3. Set the macro values in the substitution file according to this snippet:

```
file "$(TEMPLATE_DIR)/evr-vmerf230.template"
{
  {
          ...
          Pul3-Delay-SP=40,
          Pul3-Width-SP=80,
          ...
          FrontOut0-Src-SP=3,
          ...
  }
}

file "$(TEMPLATE_DIR)/evr-pulserMap.template"{
pattern { PID    F,          EVT,  ID}
                        ...
     { 3,     Trig,      4,    0 }
                        ...
}
```

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

5. **Install the** prepared **IOC** by running command `swit -V` from your project folder *MTEST-VME-EVRTEST*.

2.2.2 Substitution snippet explanation:

First we set up the pulse generator:

- **Pul3-Delay-SP** [ns]: Set the delay from when the event occurs to the start of the pulse (pulse rising edge) for pulser 3.
- **Pul3-Width-SP** [ns]: Set the pulse width (time between the pulse rising and falling edge) for pulser 3.

Next, we use the macro **FrontOut0-Src-SP** to set the source of the front panel output 0 to pulser 3 through a mapping. Mappings 0-15 correspond to pulsers 0-15. A complete list of mappings is available in the EVR manual [1].

Finally set the Pulser 3 to trigger on reception of the event 4:

- **PID**: Select Pulser 3
- **F**: Select the *Trigger* function of the pulser
- **EVT**: Map Pulser 3 Trig function to event 4
- **ID**: Unique ID for each PID-F combination.

In order to use different pulser simply change the pulser number, eg. using `Pul5-Delay-SP` instead of `Pul3-Delay-SP` sets the delay of pulser 5 instead of pulser 3. Similar is for outputs, eg. using `FrontOut1-Src-SP` instead of `FrontOut0-Src-SP` sets the output source signal of front panel output 1 instead of front panel output 0.

2.3 Process an EPICS record uppon receiving an event

This tutorial demonstrates how to use macro substitutions to trigger an EPICS event number 1 and increase a counter uppon each reception of event 1 from the timing network. The counter is a calc record, that counts how many times the event 1 was received from the timing system. Its name is in form of `$(SYS)-$(EVR):Event-$(EVT)-Cnt-I`, where `$(SYS)` represents the system name (set in the startup script), `$(EVR)` represents the Event Receiver name (set in the startup script, default "EVR0") and `$(EVT)` is the event from the timing network (in this case "1"), thus the full **name of the counter record is** `MTEST-VME-EVRTEST-EVR0:Event-1-Cnt-I`.

2.3.1 Instructions:

1. Create a **project folder**, eg. `MTEST-VME-EVRTEST`.
2. Copy **startup script** and an example **substitution file** to your project. Use the following commands in your project folder:

```
        cp /fin/devl/iocBoot/templates/slejko/EVR_VME.startup
    MTEST-VME-EVRTEST_startup.script
        cp /fin/devl/iocBoot/templates/slejko/EVR.subs cfg/
```

3. Set the macro values in the substitution file according to this snippet:

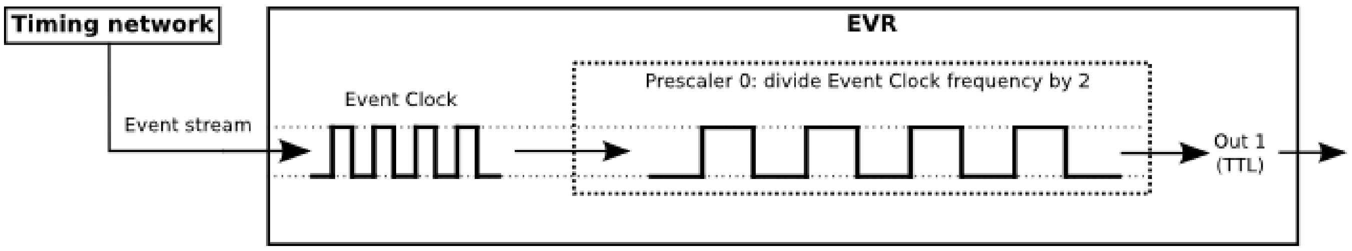
```
file "$(TEMPLATE_DIR)/evr-softEvent.template"{
pattern { EVT,   CODE }
        { "1",     "1"}
                   ...
}
```

4. Optionally, you can remove all the macros whose values you did not change.
5. **Install the** prepared **IOC** by running command `swit -V` from your project folder *MTEST-VME-EVRTEST*.

2.3.2 Substitution snippet explanation:

- **EVT**: Set to event received from timing network (hardware).
- **CODE**: Set to EPICS event number (software). It is suggested that macros *EVT* and *CODE* are set to the same value for simplicity, allthough this is not mandatory.

2.4 Generate a clock signal



Event Receivers have synchronized event clock across the timing system (the same phase and frequency). The event clock can be prescaled and mapped to the EVR output. In order to achieve this, the macro `PS0-Div-SP` is set to 2, which configures the prescaler 0(PS0) to divide the event clock frequency by 2. Then the value of the output source macro `FrontOut1-Src-SP` is set to 40, which configures the front panel output 1(FrontOut1) to use prescaler 0 as its source.

2.4.1 Instructions:

1. Create a **project folder**, eg. `MTEST-VME-EVRTEST`.
2. Copy **startup script** and an example **substitution file** to your project. Use the following commands in your project folder:

```
        cp /fin/devl/iocBoot/templates/slejko/EVR_VME.startup
MTEST-VME-EVRTEST_startup.script
        cp /fin/devl/iocBoot/templates/slejko/EVR.subs cfg/
```

3. Set the macro values in the substitution file according to this snippet:

```
file "$(TEMPLATE_DIR)/evr-vmerf230.template"
{
        {
                ...
                PS0-Div-SP=2,
                ...
                FrontOut1-Src-SP=40,
                ...
        }
}
```

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

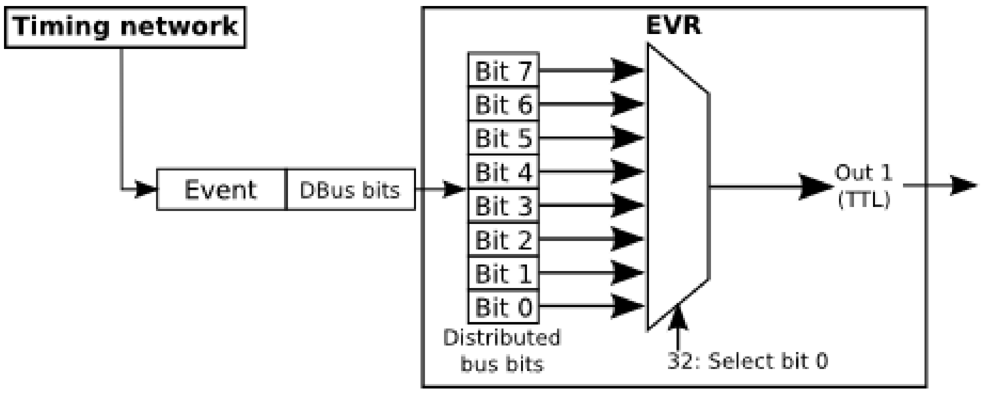
5. **Install the** prepared **IOC** by running command `swit -V` from your project folder *MTEST-VME-EVRTEST*.

2.4.2 Substitution snippet explanation:

- **PS0-Div-SP**: Set the Prescaler 0 to divide event clock frequency by 2.
- **FrontOut1-Src-SP**: Set the source of the Front Panel Output 1 to Prescaler 0 through a mapping. Mappings 40-42 correspond to prescalers 0-2. A complete list of mappings is available in the EVR manual [1].

In order to use different prescaler, simply change the prescaler number, eg. using `PS2-Div-SP` instead of `PS0-Div-SP` sets the divider of prescaler 2 instead of prescaler 0. Similar is for outputs, eg. using `FrontOut0-Src-SP` instead of `FrontOut1-Src-SP` sets the output source signal of front panel output 0 instead of front panel output 1.

2.5 Output a Distributed Bus bit



This tutorial demonstrates how to set up the DBus bit as a source of an EVR front panel output. To achieve this, the front panel output 0 should be mapped to the distributed bus bit (signal) 0. This is configured by setting the value of the output source macro `FrontOut1-Src-Sp` to 32. Mapping 32 corresponds to the DBus bit 0.

2.5.1 Instructions:

1. Create a **project folder**, eg. `MTEST-VME-EVRTEST`.

2. Copy **startup script** and an example **substitution file** to your project. Use the following commands in your project folder:

```
        cp /fin/devl/iocBoot/templates/slejko/EVR_VME.startup
  MTEST-VME-EVRTEST_startup.script
        cp /fin/devl/iocBoot/templates/slejko/EVR.subs cfg/
```

3. Set the macro values in the substitution file according to this snippet:

```
file "$(TEMPLATE_DIR)/evr-vmerf230.template"
{
        {
                ...
                FrontOut1-Src-SP=32,
                ...
                }
        }
```

4. Optionally, you can remove all the macros whose values you did not change.
5. **Install the** prepared **IOC** by running command `swit -V` from your project folder *MTEST-VME-EVRTEST*.

2.5.2 Substitution snippet explanation:

- **FrontOut1-Src-SP**: Set the source of the front panel output 1 to DBus bit 0 through a mapping. Mappings 32-39 correspond to DBus bits 0-7. A complete list of mappings is available in the EVR manual [1].

In order to use different front panel output, simply change the front panel output number, eg. using `FrontOut0-Src-SP` instead of `FrontOut1-Src-SP` sets the output source signal of front panel output 0 instead of front panel output 1.

References

[1] Evr manual. `https://github.psi.ch/projects/ED/repos/mrfioc2/browse/documentation/PSI/evr_manual.pdf`.

[2] Substitution file. `https://github.psi.ch/projects/ED/repos/mrfioc2/browse/PSI/evr_ex.subs?raw`.

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

[4] PSI. How to set up IFC 1210. `https://controls.web.psi.ch/cgi-bin/twiki/view/Main/HowToSetupIFC1210ioc`.