Micro-Research Finland Oy
Jorvas Hitech Center, Hirsalantie 11, 02420 Jorvas, Finland

Document: EVR-MRM-001.doc
Date: 30 June 2010
Issue: 1
Page: 1 of 60
Author: Jukka Pietarinen

Event Receiver

cPCI-EVR-220, cPCI-EVR-230, PMC-EVR-230, VME-EVR-230, VME-EVR-230RF, cPCI-EVR-TG-300, cRIO-EVR-300 and cPCI-EVR-300

Technical Reference

Firmware Version 0004

Contents

Introduction

Event Receivers decode timing events and signals from an optical event stream transmitted by an Event Generator. Events and signals are received at predefined rate the event clock that is usually divided down from an accelerators main RF reference. The event receivers lock to the phase event clock of the Event Generator and are thus phase locked to the RF reference. Event Receivers convert event codes transmitted by an Event Generator to hardware outputs. They can also generate software interrupts and store the event codes with globally distributed timestamps into FIFO memory to be read by a CPU.

Functional Description

After recovering the event clock the Event Receiver demultiplexes the event stream to 8-bit distributed bus data and 8-bit event codes. The distributed bus may be configured to share its bandwidth with time deterministic data transmission.

Event Decoding

The Event Receiver provides two mapping RAMs of 8 × 128 bits. Only one of the RAMs can be active at a time, however both RAMs may be modified at any time. The event code is applied to the address lines of the active mapping RAM. The 128-bit data programmed into a specific memory location pointed to by the event code determines what actions will be taken.

Event code	Offset	Internal functions	Pulse Triggers	'Set' Pulse	'Reset' Pulse
0x00	0x0000	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits
0x01	0x0010	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits
0x02	0x0020	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits
…	…	…	…	…	…
0xFF	0x0FF0	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits

There are 32 bits reserved for internal functions which are by default mapped to the event codes shown in table . The remaining 96 bits control internal pulse generators. For each pulse generator there is one bit to trigger the pulse generator, one bit to set the pulse generator output and one bit to clear the pulse generator output.

Map bit	Default event code	Function
127	n/a	Save event in FIFO
126	n/a	Latch timestamp
125	n/a	Led event
124	n/a	Forward event from RX to TX
123	0x79	Stop event log
122	n/a	Log event
102 to 121	n/a	(Reserved)
101	0x7a	Hearbeat
100	0x7b	Reset Prescalers
99	0x7d	Timestamp reset event
98	0x7c	Timestamp clock event
97	0x71	Seconds shift register '1'
96	0x70	Seconds shift register '0'

74 to 95	(Reserved)
73	Trigger pulse generator 9
…	…
64	Trigger pulse generator 0
42 to 63	(Reserved)
41	Set pulse generator 9 output high
…	…
32	Set pulse generator 0 output high
10 to 31	(Reserved)
9	Reset pulse generator 9 output low
…	…
0	Reset pulse generator 0 output low

Heartbeat Monitor

A heartbeat monitor is provided to receive heartbeat events. Event code $7A is by default set up to reset the heartbeat counter. If no heartbeat event is received the counter times out (approx. 1.6 s) and a heartbeat flag is set. The Event Receiver may be programmed to generate a heartbeat interrupt.

Event FIFO and Timestamp Events

The Event System provides a global timebase to attach timestamps to collected data and performed actions. The time stamping system consists of a 32-bit timestamp event counter and a 32-bit seconds counter. The timestamp event counter either counts received timestamp counter clock events or runs freely with a clock derived from the event clock. The event counter is also able to run on a clock provided on a distributed bus bit.

The event counter clock source is determined by the prescaler control register. The timestamp event counter is cleared at the next event counter rising clock edge after receiving a timestamp event counter reset event. The seconds counter is updated serially by loading zeros and ones (see mapping register bits) into a shift register MSB first. The seconds register is updated from the shift register at the same time the timestamp event counter is reset.

The timestamp event counter and seconds counter contents may be latched into a timestamp latch. Latching is determined by the active event map RAM and may be enabled for any event code.

An event FIFO memory is implemented to store selected event codes with attached timing information. The 80-bit wide FIFO can hold up to 511 events. The recorded event is stored along with 32-bit seconds counter contents and 32-bit timestamp event counter contents at the time of reception. The event FIFO as well as the timestamp counter and latch are accessible by software.

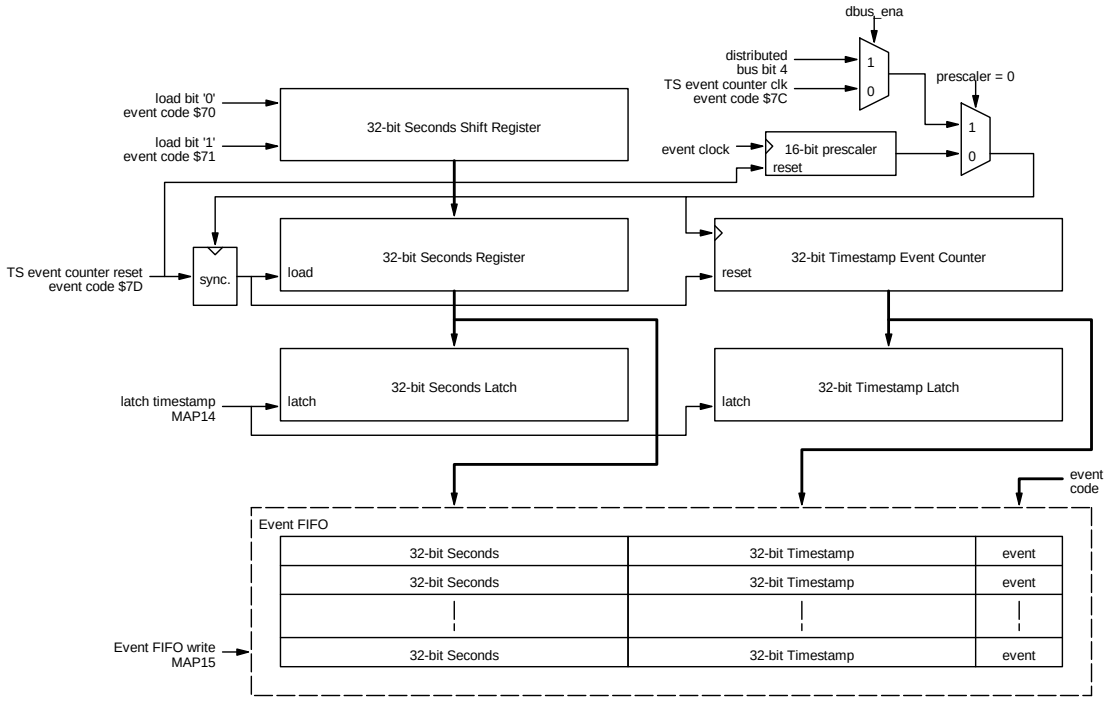

Figure 1: Event FIFO and Timestamping

Event Log

Up to 512 events with timestamping information can be stored in the event log. The log is implemented as a ring buffer and is accessible as a memory region. Logging events can be stopped by an event or software.

Distributed Bus and Data Transmission

The distributed bus is able to carry eight simultaneous signals sampled with the event clock rate over the fibre optic transmission media. The distributed bus signals may be output on programmable front panel outputs.

The 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. The remaining bandwidth is reserved for transmitting data with a speed up to 50 Mbytes/s (event clock rate divide by two).

Pulse Generators

The structure of the pulse generation logic is shown in Figure 2. Three signals from the mapping RAM control the output of the pulse: trigger, ‘set’ pulse and ‘reset’ pulse. A *trigger* causes the delay counter to start counting, when the end-of-count is reached the output pulse changes to the ‘set’ state and the width counter starts counting. At the end of the width count the output pulse is cleared. The mapping RAM signal ‘set’ and ‘reset’ cause the output to change state immediately without any delay.

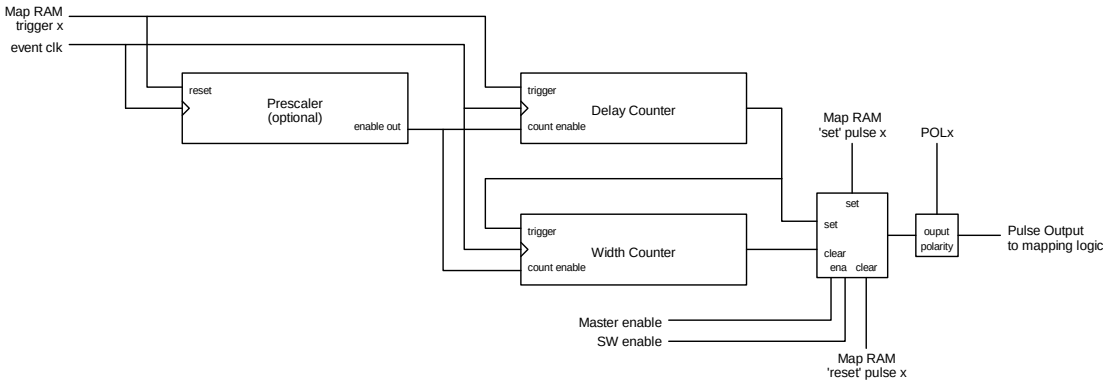

Figure 2: Pulse Output Structure

32 bit registers are reserved for both counters and the prescaler, however, the prescaler is not necessarily implemented for all channels and may be hard coded to 1 in case the prescaler is omitted. Software may write 0xFFFFFFFF to these registers and read out the actual width or hard-coded value of the register. For example if the width counter is limited to 16 bits a read will return 0x0000FFFF after a write of 0xFFFFFFFF.

Prescalers

The Event Receiver provides a number of programmable prescalers. The frequencies are programmable and are derived from the event clock. A special event code reset prescalers $7B causes the prescalers to be synchronously reset, so the frequency outputs will be in same phase across all event receivers.

Programmable Front Panel Connections

The front panel outputs are programmable: each pulse generator output, prescaler and distributed bus bit can be mapped to any output. The mapping is shown in table below.

Table 1: Signal mapping IDs

Mapping ID	Signal
0	Pulse generator 0 output
…	…
9	Pulse generator 9 output
10 to 31	(Reserved)
32	Distributed bus bit 0 (DBUS0)
…	…
39	Distributed bus bit 7 (DBUS7)
40	Prescaler 0
41	Prescaler 1
42	Prescaler 2
43 to 61	(Reserved)
62	Force output high (logic 1)
63	Force output low (logic 0)

Front Panel TTL Outputs (VME-EVR-230 and VME-EVR-230RF)

The VME-EVR-230 provides eight programmable TTL outputs in the front panel TTL0 to TTL7 whereas the number of TTL level outputs in the VME-EVR-230RF is limited to four (TTL0 to TTL3). These outputs are capable of driving a TTL level signal into a 50 ohm ground terminated coaxial cable. The source for these signals are determined by mapping registers which allow selecting different types of pulse outputs, prescalers and distributed bus signals.

Front Panel Universal I/O Slots

Universal I/O slots provide different types of output with exchangeable Universal I/O modules. Each module provides two outputs e.g. two TTL output, two NIM output or two optical outputs. The source for these outputs is selected with mapping registers.

The two front panel Universal I/O slots have extra I/O pins to allow controlling the delay of UNIV-LVPECL-DLY modules.

An optional side-by-side front panel module for the cPCI-EVR-220 and cPCI-EVR-230 offers three additional Universal I/O slots with a maximum of six outputs.

Front Panel CML Outputs (VME-EVR-230RF only)

Front Panel CML Outputs provide low jitter differential signals with special outputs. The outputs can work in different configurations: pulse mode, pattern mode and frequency mode.

CML Pulse Mode

The source for these outputs is selected in a similar way than the TTL outputs using mapping registers, however, the output logic monitors the state of this signal and distinguishes between state low (00), rising edge (01), high state (11) and falling edge (10). Based on the state a 20 bit pattern is sent out with a bit rate of 20 times the event clock rate.

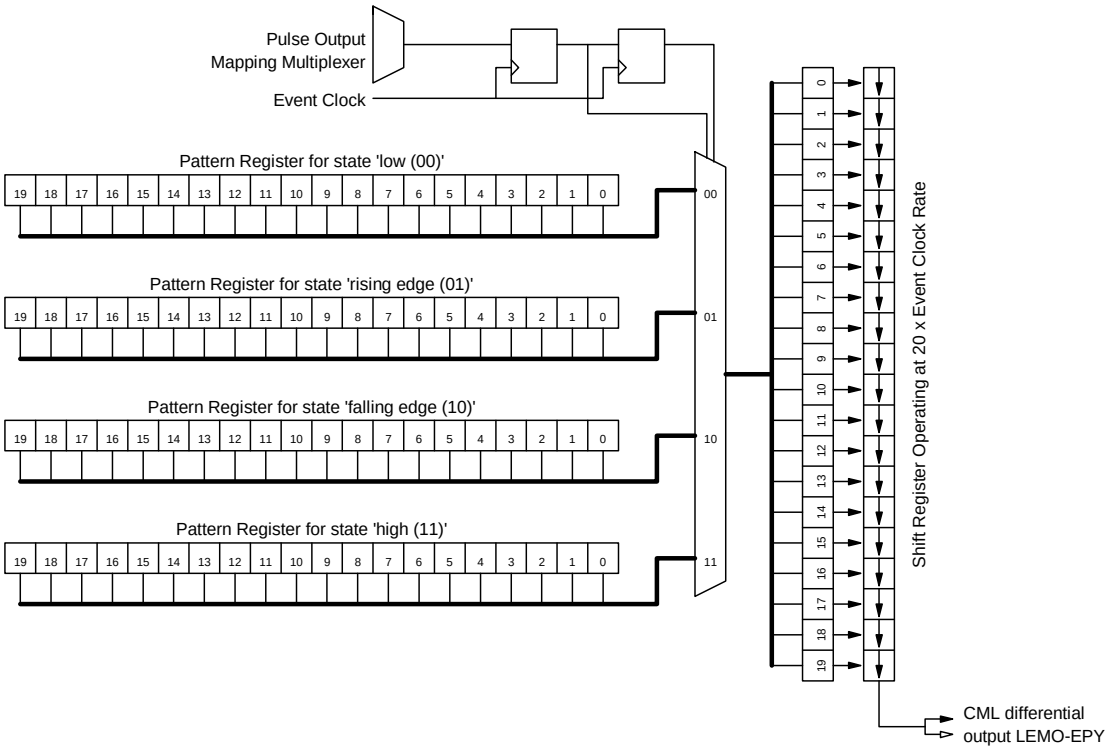

Figure 3: Block Diagram of Programmable CML Outputs

- When the source for a CML output is low and was low one event clock cycle earlier (state low), the CML output repeats the 20 bit pattern stored in pattern_00 register.
- When the source for a CML output is high and was low one event clock cycle earlier (state rising), the CML output sends out the 20 bit pattern stored in pattern_01 register.
- When the source for a CML output is high and was high one event clock cycle earlier (state high), the CML output repeats the 20 bit pattern stored in pattern_11 register.
- When the source for a CML output is low and was high one event clock cycle earlier (state falling), the CML output sends out the 20 bit pattern stored in pattern_10 register.

For an event clock of 125 MHz the duration of one single CML output bit is 400 ps. These outputs allow for producing fine grained adjustable output pulses and clock frequencies.

CML Frequency Mode

In frequency mode one can generate clocks where the clock period can be defined in steps of $1/20^{th}$ part of the event clock cycle i.e. 400 ps step with an event clock of 125 MHz. There are some limitations, however:

- Clock high time and clock low time must be $\geq 20/20^{th}$ event clock period steps
- Clock high time and clock low time must be $< 65536/20^{th}$ event clock period steps

The clock output can be synchronized by one of the pulse generators, distributed bus signal etc. When a rising edge of the mapped output signal is detected the frequency generator takes its

output value from the trigger level bit and the counter value from the trigger position register. Thus one can adjust the phase of the synchronized clock in 1/20th steps of the event clock period.

Usage example: Australian synchrotron booster clock. We have following:

- Event clock of 499.654 MHz/4
- Storage ring 360 RF buckets
- Booster 217 RF buckets
- Booster and storage ring coincidence clock on DBUS7

The CML outputs are running at a rate of 20 times the event clock or 499.654 MHz * 5, thus the booster revolution period is 217 * 5 CML bit periods. In CML frequency mode we can now set the output period (pulse high time + pulse low time) to 217 * 5 = 1085 bits. For approximately 50% duty cycle we set the pulse high time to 542 (0x21e) and the pulse low time to 543 (0x21f).

The actual register settings required are:
Write 0x00000011 to CML Control register (CMLxENA)
Write 0x021e to CML High Period Count register (CMLxHP)
Write 0x021f to CML Low Period Count register (CMLxLP)

We also need to set the trigger from DBUS7 by setting up register FPOutMapx.

To change the generated clock phase in respect to the trigger we can select the trigger polarity by bit CMLTL in the CML Control register and the trigger position also in the CML Control register.

CML Pattern Mode

In pattern mode one can generate arbitrary bit patterns taking into account following:

- The pattern length is a multiple of 20 bits, where each bit is 1/20th of the event clock period
- Maximum length of the arbitrary pattern is 20 × 2048 bits
- A pattern can be triggered from any pulse generator, distributed bus bit etc. When triggered the pattern generator starts sending 20 bit words from the pattern memory sequentially starting from position 0. This goes on until the pattern length set by the samples register has been reached.
- If the pattern generator is in recycle mode the pattern continues immediately from position 0 of the pattern memory.
- If the pattern generator is in single pattern mode, the pattern stops and the 20 bit word from the last position of the pattern memory (2047) is sent out until the pattern generator is triggered again.

cPCI-EVRTG-300 GTX Front Panel Outputs

All eight cPCI-EVRTG-300 front panel output are similar to the CML outputs on the VME-EVR-230RF. The GTX Outputs provide low jitter differential signals with special outputs. The outputs can work in different configurations: pulse mode, pattern mode and frequency mode. The difference compared to the CML output of the VME-EVR-230RF is that instead of 20 bits per

event clock cycle the GTX outputs have 40 bits per event clock cycle doubling the resolution to 200 ps/bit at an event clock of 125 MHz.

GTX Pulse Mode

The source for these outputs is selected in a similar way than the TTL outputs using mapping registers, however, the output logic monitors the state of this signal and distinguishes between state low (00), rising edge (01), high state (11) and falling edge (10). Based on the state a 40 bit pattern is sent out with a bit rate of 40 times the event clock rate.

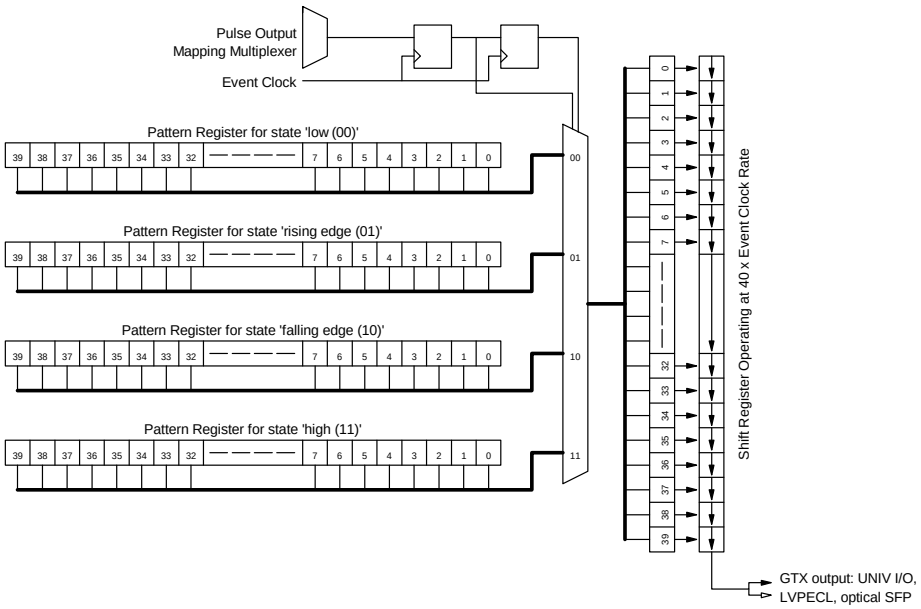

Figure 4: Block Diagram of Programmable GTX Outputs

- When the source for a GTX output is low and was low one event clock cycle earlier (state low), the GTX output repeats the 40 bit pattern stored in pattern_00 register.
- When the source for a GTX output is high and was low one event clock cycle earlier (state rising), the GTX output sends out the 40 bit pattern stored in pattern_01 register.
- When the source for a GTX output is high and was high one event clock cycle earlier (state high), the GTX output repeats the 40 bit pattern stored in pattern_11 register.
- When the source for a GTX output is low and was high one event clock cycle earlier (state falling), the GTX output sends out the 40 bit pattern stored in pattern_10 register.

For an event clock of 125 MHz the duration of one single GTX output bit is 200 ps. These outputs allow for producing fine grained adjustable output pulses and clock frequencies.

GTX Frequency Mode

In frequency mode one can generate clocks where the clock period can be defined in steps of $1/40^{th}$ part of the event clock cycle i.e. 200 ps step with an event clock of 125 MHz. There are some limitations, however:

- Clock high time and clock low time must be ≥ 40/40th event clock period steps
- Clock high time and clock low time must be < 65536/40th event clock period steps

The clock output can be synchronized by one of the pulse generators, distributed bus signal etc. When a rising edge of the mapped output signal is detected the frequency generator takes its output value from the trigger level bit and the counter value from the trigger position register. Thus one can adjust the phase of the synchronized clock in 1/40th steps of the event clock period.

To change the generated clock phase in respect to the trigger we can select the trigger polarity by bit CMLTL in the CML Control register and the trigger position also in the CML Control register.

GTX Pattern Mode

In pattern mode one can generate arbitrary bit patterns taking into account following:

- The pattern length is a multiple of 40 bits, where each bit is 1/40th of the event clock period
- Maximum length of the arbitrary pattern is 40 × 2048 bits
- A pattern can be triggered from any pulse generator, distributed bus bit etc. When triggered the pattern generator starts sending 40 bit words from the pattern memory sequentially starting from position 0. This goes on until the pattern length set by the samples register has been reached.
- If the pattern generator is in recycle mode the pattern continues immediately from position 0 of the pattern memory.
- If the pattern generator is in single pattern mode, the pattern stops and the 40 bit word from the last position of the pattern memory (2047) is sent out until the pattern generator is triggered again.

Configurable Size Data Buffer

Some applications require deterministic data transmission. The configurable size data buffer provides a configurable size buffer that may be transmitted over the event system link. The buffer size is configured in the Event Generator to up to 2 kbytes. The Event Receiver is able to receive buffers of any size from 4 bytes to 2 kbytes in four byte (long word) increments.

Data reception is enabled by changing the distributed bus mode for data transmission (*mode* = 1 in Data Buffer Control Register). This halves the distributed bus update rate. Before a data buffer can be received the data buffer receiver has to be enabled (write *enable* = 1 in control register). This clears the checksum error flag and sets the rx_enable flag. When a data buffer has been received the rx_enable flag is cleared and rx_complete flag is set. If the received and computed checksums do not match the checksum error flag is set.

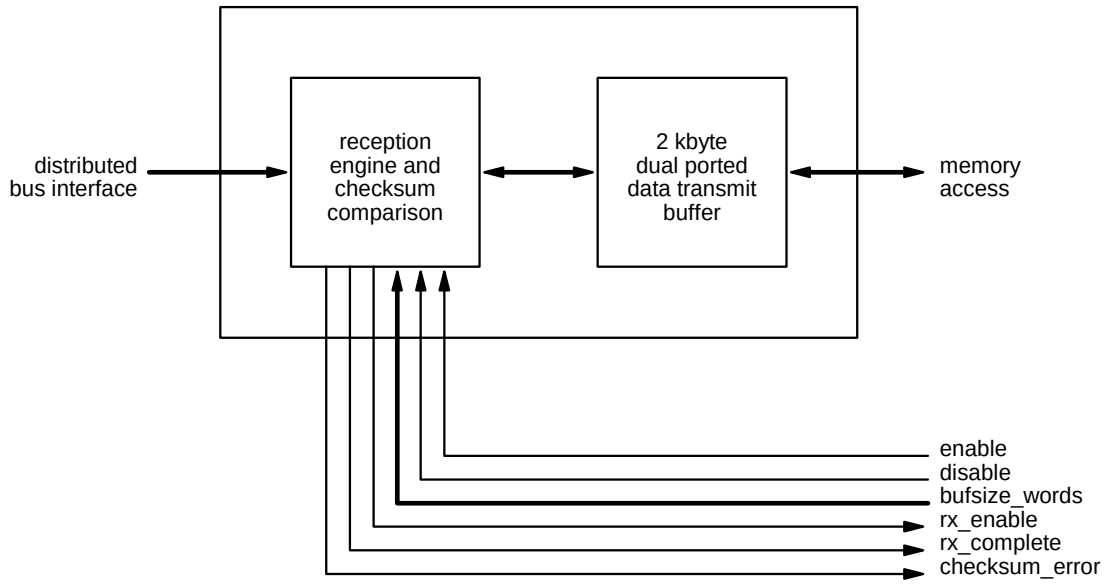

Figure 5: Data Receive Buffer

The size of the data buffer transfer can be read from the control register. An interrupt may be generated after reception of a data buffer.

Interrupt Generation

The Event Receiver has multiple interrupt sources which all have their own enable and flag bits. The following events may be programmed to generate an interrupt:

- Receiver link state change
- Receiver violation: bit error or the loss of signal.
- Lost heartbeat: heartbeat monitor timeout.
- Write operation of an event to the event FIFO.
- Event FIFO is full.
- Data Buffer reception complete.

In addition to the events listed above an interrupt can be generated from one of the pulse generator outputs, distributed bus bits or prescalers. The pulse interrupt can be mapped in a similar way as the front panel outputs.

External Event Input

An external hardware input is provided to be able to take an external pulse to generate an internal event. This event will be handled as any other received event.

Programmable Reference Clock

The event receiver requires a reference clock to be able to synchronise 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.

Fractional Synthesiser

The clock reference for the event receiver is generated on-board the event receiver using a fractional synthesiser. A Micrel (http://www.micrel.com) SY87739L Protocol Transparent Fractional-N Synthesiser with a reference clock of 24 MHz is used. The following table lists programming bit patterns for a few frequencies.

Event Rate	Configuration Bit Pattern	Reference Output	Precision (theoretical)
499.8 MHz/5 = 99.96 MHz	0x025B41ED	99.956 MHz	-40 ppm
50 MHz	0x009743AD	50.0 MHz	0
499.8 MHz/10 = 49.98 MHz	0x025B43AD	49.978 MHz	-40 ppm

The event receiver reference clock is required to be in ±100 ppm range of the event generator event clock.

Connections

cPCI-EVR-2x0 Front Panel Connections

The front panel of the Event Receiver and its optional side-by-side module is shown in Figure 6 and Figure 7.

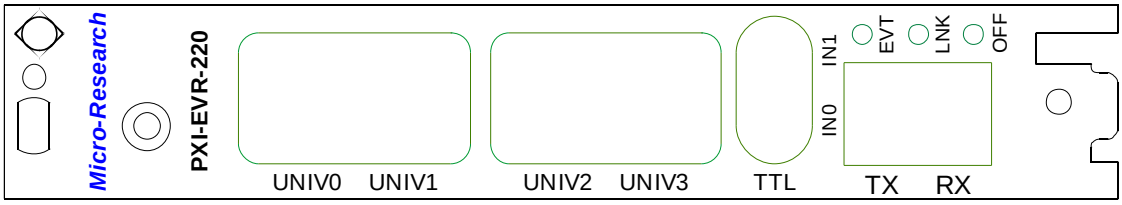

Figure 6: Event Receiver Front Panel

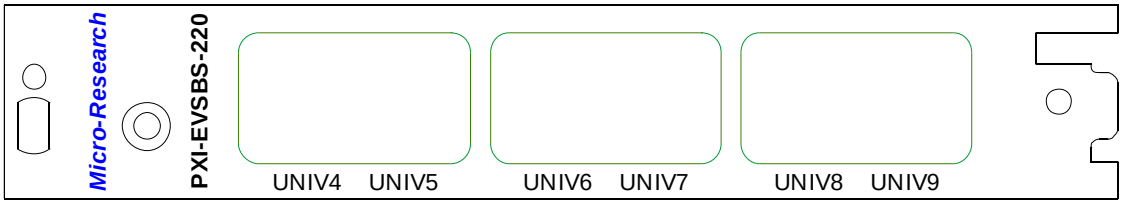

Figure 7: Optional Side-by-side Module Front Panel

The front panel of the Event Receiver includes the following connections and status leds:

Connector / Led	Style	Level	Description
LNK	Red/Green Led		Red: receiver violation detected Green: RX link OK, violation flag

			cleared
EVT	Red/Green Led		Green: link OK, flashes when event code received Red: Flashes on led event
TX	LC	optical	Transmit Optical Output (TX)
RX	LC	optical	Receiver Optical Input (RX)
TTL IN0	LEMO-EPY	TTL	External Event Input
TTL IN1	LEMO-EPY	TTL	External Event Input
UNIV0/1	Universal slot		Universal Output 0/1
UNIV2/3	Universal slot		Universal Output 2/3
UNIV4/5	Universal slot		Universal Output 4/6
UNIV6/7	Universal slot		Universal Output 6/7
UNIV8/9	Universal slot		Universal Output 8/9

VME-EVR-230 and VME-EVR-230RF Front Panel Connections

The front panel of the VME-EVR-230 Event Receiver is shown in Figure 6 and VME-EVR-230RF in Figure 9: VME-EVR-230RF Event Receiver Front PanelFigure 9 respectively.

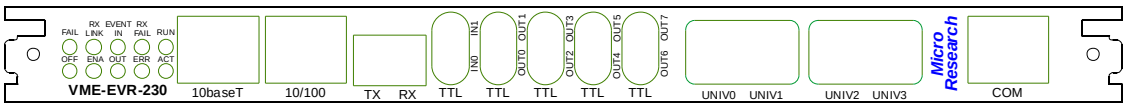

Figure 8: VME-EVR-230 Event Receiver Front Panel

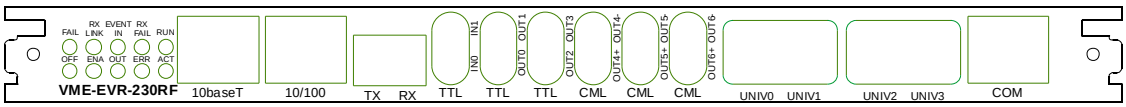

Figure 9: VME-EVR-230RF Event Receiver Front Panel

The front panel of the Event Receiver includes the following connections and status leds:

Connector / Led	Style	Level	Description
FAIL	Red Led		Module Failure/Interlock active
OFF	Blue Led		Module not Configured/Powered Down
RX LINK	Green Led		Receiver Link Signal OK
ENA	Green Led		Event Receiver Enabled
EVENT IN	Yellow Led		Incoming Event (RX)
EVENT OUT	Yellow Led		Active HW output
RX FAIL	Red Led		Receiver Violation
ERR	Red Led		SY87739L reference not locked
RUN	Green Led		Ubicom IP2022 software running
ACT	Yellow Led		Ubicom IP2022 telnet connection active
10baseT with LEDs	RJ45 green Led amber Led	10baseT	10baseT Ethernet Connection link established link activity

10/100	RJ45		(reserved)
TX	LC	optical	Transmit Optical Output (TX)
RX	LC	optical	Receiver Optical Input (RX)
TTL IN0	LEMO-EPY	TTL	External Event Input
TTL IN1	LEMO-EPY	TTL	External Event Input
TTL OUT0	LEMO-EPY	TTL	Programmable TTL Output 0
TTL OUT1	LEMO-EPY	TTL	Programmable TTL Output 1
TTL OUT2	LEMO-EPY	TTL	Programmable TTL Output 2
TTL OUT3	LEMO-EPY	TTL	Programmable TTL Output 3
TTL OUT4	LEMO-EPY	TTL	Programmable TTL Output 4[1]
TTL OUT5	LEMO-EPY	TTL	Programmable TTL Output 5
TTL OUT6	LEMO-EPY	TTL	Programmable TTL Output 6
TTL OUT7	LEMO-EPY	TTL	Programmable TTL Output 7
CML OUT4	LEMO-EPY	CML	Programmable CML Output 4[2]
CML OUT5	LEMO-EPY	CML	Programmable CML Output 5
CML OUT6	LEMO-EPY	CML	Programmable CML Output 6
UNIV0/1	Universal slot		Universal Output 0/1
UNIV2/3	Universal slot		Universal Output 2/3
COM	RJ45	RS232	(reserved)

VME P2 User I/O Pin Configuration

The following table lists the connections to the VME P2 User I/O Pins.

Pin	Signal
A1	Transition board ID0
A2	Transition board ID1
A3-A10	Ground
A11	Transition board ID2
A12	Transition board ID3
A13-A15	Ground
A16	Transition board handle switch
A17-A26	Ground
A27-A31	+5V
A32	Power control for transition board
C1	(reserved)
C2	(reserved)
C3	(reserved)
C4	(reserved)
C5	(reserved)
C6	(reserved)
C7	(reserved)
C8	(reserved)
C9	(reserved)

[1] TTL outputs TTL4-TTL7 available on VME-EVR-230 only
[2] CML outputs available on VME-EVR-230RF only

C10	(reserved)
C11	(reserved)
C12	Programmable transition board output 0
C13	Programmable transition board output 1
C14	Programmable transition board output 2
C15	Programmable transition board output 3
C16	Programmable transition board output 4
C17	Programmable transition board output 5
C18	Programmable transition board output 6
C19	Programmable transition board output 7
C20	Programmable transition board output 8
C21	Programmable transition board output 9
C22	Programmable transition board output 10
C23	Programmable transition board output 11
C24	Programmable transition board output 12
C25	Programmable transition board output 13
C26	Programmable transition board output 14
C27	Programmable transition board output 15
C28	(reserved)
C29	(reserved)
C30	(reserved)
C31	(reserved)
C32	(reserved)

PMC-EVR-230 Front Panel Connections

The front panel of the PMC Event Receiver is shown in Figure 10.

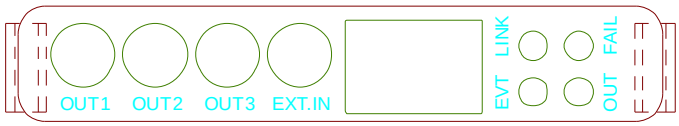

Figure 10: PMC-EVR-230 Event Receiver Front Panel

The front panel of the Event Receiver includes the following connections and status leds:

Connector / Led	Style	Level	Description
LINK	Green Led		Receiver Link Signal OK
EVT	Yellow Led		Incoming Event (RX)
OUT	Yellow Led		Active HW output
FAIL	Red Led		Receiver Violation
TX (SFP) next to leds	LC	Optical 850 nm	Event link Transmit
RX (SFP) next to EXT.IN	LC	Optical 850 nm	Event link Receiver
OUT0	LEMO-EPL	TTL	Programmable TTL Output 0

OUT1	LEMO-EPL	TTL	Programmable TTL Output 1
OUT2	LEMO-EPL	TTL	Programmable TTL Output 2
EXT IN	LEMO-EPL	TTL	External Event Input

PMC-EVR-230 Pn4 User I/O Pin Configuration

The following table lists the connections to the PMC Pn4 User I/O Pins and to VME P2 pins when the module is mounted on a host with "P4V2-64ac" mapping complying VITA-35 PMC-P4 to VME-P2-Rows-A,C.

PMC Pn4 pin	VME P2 Pin	Signal
2	A1	Transition board ID0
4	A2	Transition board ID1
6, 8, …, 20	A3-A10	Ground
22	A11	Transition board ID2
24	A12	Transition board ID3
26, 28, 30	A13-A15	Ground
32	A16	Transition board handle switch
34, 36, …, 52	A17-A26	Ground
54, 56, …, 62	A27-A31	+5V
64	A32	Power control for transition board
1	C1	(reserved)
3	C2	(reserved)
5	C3	(reserved)
7	C4	(reserved)
9	C5	(reserved)
11	C6	(reserved)
13	C7	(reserved)
15	C8	(reserved)
17	C9	(reserved)
19	C10	(reserved)
21	C11	(reserved)
23	C12	Programmable transition board output 0
25	C13	Programmable transition board output 1
27	C14	Programmable transition board output 2
29	C15	Programmable transition board output 3
31	C16	Programmable transition board output 4
33	C17	Programmable transition board output 5
35	C18	Programmable transition board output 6
37	C19	Programmable transition board output 7
39	C20	Programmable transition board output 8
41	C21	Programmable transition board output 9
43	C22	Programmable transition board output 10

45	C23	Programmable transition board output 11
47	C24	Programmable transition board output 12
49	C25	Programmable transition board output 13
51	C26	Programmable transition board output 14
53	C27	Programmable transition board output 15
55	C28	(reserved)
57	C29	(reserved)
59	C30	(reserved)
61	C31	(reserved)
63	C32	(reserved)

cRIO-EVR-300 Front Panel Connections

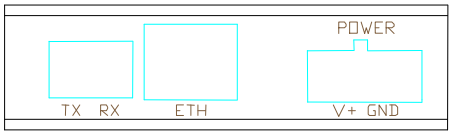

Figure 11: cRIO-EVR-300 Event Receiver Front Panel

Connector / Led	Style	Level	Description
TX (SFP)	LC	Optical 850 nm	Event link Transmit
RX (SFP)	LC	Optical 850 nm	Event link Receiver
ETH	RJ45	10baseT/100baseTX	Ethernet port
V+	Terminal	+6 to +30 VDC	Power supply positive supply
GND	Terminal	Ground	Power supply ground

cPCI-EVRTG-300 Front Panel Connections

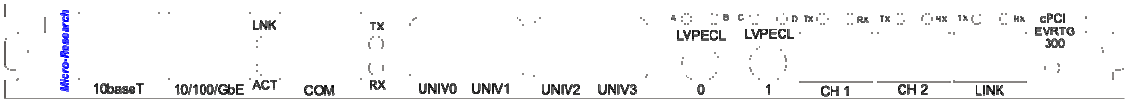

Figure 12: cPCI-EVRTG-300 Event Receiver Front Panel

Connector / Led	Style	Level	Description
10baseT with LEDs	RJ45 green Led amber Led	10baseT	10baseT Ethernet Connection link established link activity
10/100/GbE	RJ45		(reserved)
LNK	led		10/100/GbE link led
ACT	led		10/100/GbE active led
COM	RJ45	RS-232	(reserved)
TX	Led		(reserved)
RX	Led		(reserved)
UNIV0/1	Universal slot		Universal Output 0/1
UNIV2/3	Universal slot		Universal Output 2/3
LVPECL 0	EPG.00.302	3.3V diff. LVPECL	LVPECL Output
LVPECL 1	EPG.00.302	3.3V diff. LVPECL	LVPECL Output

A	RGB Led		(reserved)
B	RGB Led		(reserved)
C	RGB Led		(reserved)
D	RGB Led		(reserved)
CH 1	LC	Optical 850 nm	GunTX Output
CH 2	LC	Optical 850 nm	GunTX Output
Link TX (SFP)	LC	Optical 850 nm	Event link Transmit
Link RX (SFP)	LC	Optical 850 nm	Event link Receiver

VME-EVR-230 and VME-EVR-230RF Network Interface

A 10baseT network interface is provided to upgrade the FPGA firmware and set up boot options. It is also possible to control the module over the network interface.

Assigning an IP Address to the Module

By default the modules uses DHCP (dynamic host configuration protocol) to acquire an IP address. In case a lease cannot be acquired the IP address set randomly in the 169.254.x.x subnet. The board can be programmed to use a static address instead if DHCP is not available.

The module can be located looking at the lease log of the DHCP server or using a Windows tool called Locator.exe.

Using Telnet to Configure Module

To connect to the configuration utility of the module issue the following command:

```
telnet 192.168.1.32 23
```

The latter parameter is the telnet port number and is required in Linux to prevent negotiation of telnet parameters which the telnet server of the module is not capable of.

The telnet server responds to the following commands:

Command	Description
b	Show/change boot parameters, IP address etc.
d	Dump 16 bytes of memory
h / ?	Show Help
m <address> [<data>]	Read/Write FPGA CR/CSR, Function 0
r	Reset Board
s	Save boot configuration & dynamic configuration values into non-volatile memory
t	Tune delay line for event clock recovery
+	Manually increase delay line delay *)
-	Manually decrease delay line delay *)
u	Update IP2022 software
q	Quit Telnet

*) This option has been added with IP2022 software version 060309 for VME-EVR-230RF (displayed in output from help command)

Boot Configuration (command b)

Command b displays the current boot configuration parameters of the module. The parameter may be changed by giving a new parameter value. The following parameters are displayed:

Parameter	Description
Use DHCP	0 = use static IP address, 1 = use DHCP to acquire address, net mask etc.
IP address	IP address of module

Subnet mask	Subnet mask of module
Default GW	Default gateway
FPGA mode	FPGA configuration mode 0 – FPGA is not configured after power up 1 – FPGA configured from internal Flash memory 2 – FPGA is configured from FTP server
FTP server	FTP server IP address where configuration bit file resides
Username	FTP server username
Password	FTP server password
FTP Filename	FTP server configuration file name
Flash Filename	Configuration file name on internal flash
µs divider	Integer divider to get from event clock to 1MHz, e.g. 125 for 124.9135 MHz
Fractional divider configuration word	Micrel SY87739UMI fractional divider configuration word to set refenrence for event clock

Note that after changing parameters the parameters have to be saved to internal flash by issuing the Save boot configuration (s) command. The changes are applied only after resetting the module using the reset command or hardware reset/power sequencing.

Memory dump (command d)

This command dumps 16 bytes of memory starting at the given address, if the address is omitted the previous address value is increased by 16 bytes.

The most significant byte of the address determines the function of the access:

Address	**Function**
0x78000000	CR/CSR space access
0x7a000000	EVR registers access

To dump the start of the EVR register map issue the ‘d’ command from the telnet prompt:

```
VME-EVR-230RF -> d 7a000000 ↵
Addr 7a000000:  1005 0001 0000 0000 0000 0000 0000 0000
VME-EVR-230RF -> d ↵
Addr 7a000010:  0000 0000 0000 0000 0000 0000 0000 0000
VME-EVR-230RF ->
```

Memory modify (commands d and m)

The access size is always a short word i.e. two bytes.

To check the status register from the telnet prompt:

```
VME-EVR-230RF -> m 7a000000 ↵
Addr 7a000000 data 1005
VME-EVR-230RF ->
```

To clear the violation flag issue:

```
VME-EVR-230RF -> m 7a000000 1005 ↵
```

```
Addr 7a000000 data 0000
VME-EVR-230RF ->
```

Tuning Delay Line (command t)

The VME Event Receiver VME-EVR-230RF has to be configured for proper event clock rate and the on-board delay line has to be tuned for the operating conditions. Before setting up the board make sure you have an Event Generator with the correct event clock connected to the Event Receiver. Also, let the EVR reach operating temperature (power on for 10 minutes in crate). See previous section for setting up the event clock rate.

To start tuning issue command 't' from the telnet prompt:

```
VME-EVR-230RF -> t ↵
Starting tuning...
Adjusted sampling phase to 75
Initial DCM phase -85
Fine tuned sampling phase to 78
Final DCM phase -73.
VME-EVR-230RF ->
```

After tuning the tuned values have to be stored in non-volatile memory:

```
VME-EVR-230RF -> s ↵
Confirm save (yes/no) ? yes ↵
Configuration saved.
VME-EVR-230RF ->
```

Upgrading IP2022 Microprocessor Software (command u)

To upgrade the Ubicom IP2022 microprocessor software download the upgrade image containing the upgrade to the module using TFTP:

Linux

In Linux use e.g. interactive tftp:

```
$ tftp 192.168.1.32
tftp> bin
tftp> put upgrade.bin /fw
tftp> quit
```

Windows

In Windows command prompt issue the following command:

```
C:\> tftp –i 192.168.1.32 PUT upgrade.bin /fw
```

When the upgrade image has been downloaded and verified, enter at the telnet prompt following:

```
VME-EVR-230 -> u ↵
Really update firmware (yes/no) ? yes ↵
```

```
Self programming triggered.
```

The Event Receiver starts programming the new software and restarts.

Upgrading FPGA Configuration File

When the FPGA configuration file resides in internal flash memory a new file system image has to be downloaded to the module. This is done using TFTP protocol:

Linux

In Linux use e.g. interactive tftp:

```
$ tftp 192.168.1.32
tftp> bin
tftp> put filesystem.bin /
tftp> quit
```

Windows

In Windows command prompt issue the following command:

```
C:\> tftp –i 192.168.1.32 PUT filesystem.bin /
```

Now the FPGA configuration file has been upgraded and the new configuration is loaded after next reset/power sequencing.

Note! Due to the UDP protocol it is recommended to verify (read back and compare) the filesystem image before restarting the module. This is done following:

Linux

In Linux use e.g. interactive tftp:

```
$ tftp 192.168.1.32
tftp> bin
tftp> get / verify.bin
tftp> quit
$ diff filesystem.bin verify.bin
$
```

If files differ you should get following message:

```
Binary files filesystem.bin and verify.bin differ
```

Windows

In Windows command prompt issue the following command:

```
C:\> tftp –i 192.168.1.32 GET / verify.bin
C:\> fc /b filesystem.bin verify.bin
Comparing files filesystem.bin and verify.bin
FC: no differences encountered
```

UDP Remote Programming Protocol

The VME-EVR can be remotely programmed using the 10baseT Ethernet interface with a protocol over UDP (User Datagram Protocol) which runs on top of IP (Internet Protocol). The default port for remote programming is UDP port 2000. The UDP commands are built upon the following structure:

access_type (1 byte)	status (1 byte)	data (2 bytes)
	address (4 bytes)	
	ref (4 bytes)	

The first field defines the access type:

access_type	Description
0x01	Read Register from module
0x02	Write and Read back Register from module

The second field tells the status of the access:

Status	Description
0	Command OK
-1	Bus ERROR (Invalid read/write address)
-2	Timeout (FPGA did not respond)
-3	Invalid command

The access size is always a short word i.e. two bytes. The most significant byte of the address determines the function of the access:

Address	**Function**
0x78000000	CR/CSR space access
0x7a000000	EVR registers access

Read Access (Type 0x01)

The host sends a UDP packet to port 2000 of the VME-EVR with the following contents:

access_type (1 byte)	status (1 byte)	data (2 bytes)
0x01	0x00	0x0000
	address (4 bytes)	
	0x7a000000 (Control and Status register Function 0 address)	
	ref (4 bytes)	
	0x00000000	

If the read access is successful the VME-EVR replies to the same host and port the message came from with the following packet:

access_type (1 byte)	status (1 byte)	data (2 bytes)
0x01	0x00	0x0032
	address (4 bytes)	
	0x7a000000 (Control and Status register Function 0 address)	

ref (4 bytes)
0x00000000

Write Access (Type 0x02)

The host sends a UDP packet to port 2000 of the VME-EVR with the following contents:

access_type (1 byte)	status (1 byte)	data (2 bytes)
0x02	0x00	0x0001

address (4 bytes)
0x7a000002 (Mapping RAM Address register Function 0 address)
ref (4 bytes)
0x00000000

If the write access is successful the VME-EVR replies to the same host and port the message came from with the following packet:

access_type (1 byte)	status (1 byte)	data (2 bytes)
0x02	0x00	0x0001

address (4 bytes)
0x80000000 (Mapping RAM Address register Function 0 address)
ref (4 bytes)
0x00000000

Notice that in the reply message the data returned really is the data read from the address specified in the address field so one can verify that the data really was written ok.

Programming Details

VME CR/CSR Support

The VME Event Receiver modules provides CR/CSR Support as specified in the VME64x specification. The CR/CSR Base Address Register is determined after reset by the inverted state of VME64x P1 connector signal pins GA4*-GA0*. In case the parity signal GAP* does not match the GAx* pins the CR/CSR Base Address Register is loaded with the value 0xf8 which corresponds to slot number 31.

Note: the boards can be used in standard VME crates where geographical pins do not exist, in this case the user may either insert jumpers to set the geographical address or use the default setting when the board's CR/CSR base address will be set to 0xf8.

After power up or reset the board responds only to CR/CSR accesses with its geographical address. Prior to accessing Event Receiver functions the board has to be configured by accessing the boards CSR space.

The Configuration ROM (CR) contains information about manufacturer, board ID etc. to identify boards plugged in different VME slots. The following table lists the required field to locate an Event Receiver module.

CR address	Register	VME-EVR-230RF
0x27, 0x2B, 0x2F	Manufacturer's ID (IEEE OUI)	0x000EB2
0x33, 0x37, 0x3B, 0x3F	Board ID	0x455246E6

For convenience functions are provided to locate VME64x capable boards in the VME crate.

```
STATUS vmeCRFindBoard(int slot, UINT32 ieee_oui, UINT32 board_id,
                      int *p_slot);
```

To locate the first Event Receiver in the crate starting from slot 1, the function has to be called following:

```
#include "vme64x_cr.h"
int slot = 1;
int slot_evr;
vmeCRFindBoard(slot, MRF_IEEE_OUI, MRF_EVR200RF_BID, &slot_evr);
```

If this function returns `OK`, an Event Receiver board was found in slot `slot_evr`.

Event Receiver Function 0,1 and 2 Registers

The Event Receiver specific register are accessed via Function 0 and Function 1 as specified in the VME64x specification. The access size for Function 0 has been limited to 2 kbytes (0x0800) so not all EVR registers are accessible through this Function. The access size for Functions 1 and 2 is 256 kbytes, so this function should not be used for A16 access. Contrary to the VME64x

specification the address/address modifier compare logic does not distinguish between privileged and non-privileged accesses and accepts both.

To enable a Function, the address decoder compare register for the Function in CSR space has to be programmed. For convenience a function to perform this is provided, too:

```
STATUS vmeCSRWriteADER(int slot, int func, UINT32 ader);
```

To configure Function 0 of a Event Receiver board in slot 3 to respond to A16 accesses at the address range 0x1800-0x1FFF the function has to be called with following values:

```
vmeCSRWriteADER(3, 0, 0x18A4);
```

ADER contents are composed of the address mask and address modifier, the above is the same as:

```
vmeCSRWriteADER(3, 0, (slot << 11) | (VME_AM_SUP_SHORT_IO << 2));
```

To get the memory mapped pointer to the configured Function 0 registers on the Event Receiver board the following VxWorks function has to be called:

```
MrfEvrStruct *pEvr;
sysBusToLocalAdrs(VME_AM_SUP_SHORT_IO, (char *) (slot << 11),
                  (void *) pEvr);
```

Note: using the data transmission capability requires more than 4 kbytes, so using function 1 with addressing mode A24 is suggested, following:

```
vmeCSRWriteADER(3, 1, (slot << 19) | (VME_AM_STD_USR_DATA << 2));
MrfEvrStruct *pEvr;
sysBusToLocalAdrs(VME_AM_STD_USR_DATA, (char *) (slot << 19),
                  (void *) pEvr);
```

Register Map

Address	Register	Type	Description
0x000	Status	UINT32	Status Register
0x004	Control	UINT32	Control Register
0x008	IrqFlag	UINT32	Interrupt Flag Register
0x00C	IrqEnable	UINT32	Interrupt Enable Register
0x010	PulseIrqMap	UINT32	Mapping register for pulse interrupt
0x020	DataBufCtrl	UINT32	Data Buffer Control and Status Register
0x024	TxDataBufCtrl	UINT32	TX Data Buffer Control and Status Register
0x02C	FWVersion	UINT32	Firmware Version Register
0x040	EvCntPresc	UINT32	Event Counter Prescaler
0x04C	UsecDivider	UINT32	Divider to get from Event Clock to 1 MHz

0x050	ClockControl	UINT32	Event Clock Control Register
0x05C	SecSR	UINT32	Seconds Shift Register
0x060	SecCounter	UINT32	Timestamp Seconds Counter
0x064	EventCounter	UINT32	Timestamp Event Counter
0x068	SecLatch	UINT32	Timestamp Seconds Counter Latch
0x06C	EvCntLatch	UINT32	Timestamp Event Counter Latch
0x070	EvFIFOSec	UINT32	Event FIFO Seconds Register
0x074	EvFIFOEvCnt	UINT32	Event FIFO Event Counter Register
0x078	EvFIFOCode	UINT16	Event FIFO Event Code Register
0x07C	LogStatus	UINT32	Event Log Status Register
0x080	FracDiv	UINT32	Micrel SY87739L Fractional Divider Configuration Word
0x088	RxInitPS	UINT32	Reserved for Initial value for RF recovery DCM phase shift (VME-EVR-230RF)
0x090	GPIODir	UINT32	Front Panel UnivIO GPIO signal direction
0x094	GPIOIn	UINT32	Front Panel UnivIO GPIO input register
0x098	GPIOOut	UINT32	Front Panel UnivIO GPIO output register
0x0A0	FineDelay0	UINT16	Fine Delay for GTX output 0 (cPCI-EVRTG-300)
0x0A2	FineDelay1	UINT16	Fine Delay for GTX output 1 (cPCI-EVRTG-300)
0x0A4	FineDelay2	UINT16	Fine Delay for GTX output 2 (cPCI-EVRTG-300)
0x0A6	FineDelay3	UINT16	Fine Delay for GTX output 3 (cPCI-EVRTG-300)
0x0A8	FineDelay4	UINT16	Fine Delay for GTX output 4 (cPCI-EVRTG-300)
0x0AA	FineDelay5	UINT16	Fine Delay for GTX output 5 (cPCI-EVRTG-300)
0x0AC	FineDelay6	UINT16	Fine Delay for GTX output 6 (cPCI-EVRTG-300)
0x0AE	FineDelay7	UINT16	Fine Delay for GTX output 7 (cPCI-EVRTG-300)
0x100	Prescaler_0	UINT32	Prescaler 0 Divider
0x104	Prescaler_1	UINT32	Prescaler 1 Divider
0x108	Prescaler_2	UINT32	Prescaler 2 Divider
0x200	Pulse0Ctrl	UINT32	Pulse 0 Control Register
0x204	Pulse0Presc	UINT32	Pulse 0 Prescaler Register
0x208	Pulse0Delay	UINT32	Pulse 0 Delay Register
0x20C	Pulse0Width	UINT32	Pulse 0 Width Register
0x210			Pulse 1 Registers
0x220			Pulse 2 Registers
…	…	…	…
0x290			Pulse 9 Registers
0x400	FPOutMap0	UINT16	Front Panel Output 0 Map Register

0x402	FPOutMap1	UINT16	Front Panel Output 1 Map Register
0x404	FPOutMap2	UINT16	Front Panel Output 2 Map Register
0x406	FPOutMap3	UINT16	Front Panel Output 3 Map Register
0x408	FPOutMap4	UINT16	Front Panel Output 4 Map Register
0x40A	FPOutMap5	UINT16	Front Panel Output 5 Map Register
0x40C	FPOutMap6	UINT16	Front Panel Output 6 Map Register
0x40E	FPOutMap7	UINT16	Front Panel Output 7 Map Register
0x440	UnivOutMap0	UINT16	Front Panel Universal Output 0 Map Register
0x442	UnivOutMap1	UINT16	Front Panel Universal Output 1 Map Register
0x444	UnivOutMap2	UINT16	Front Panel Universal Output 2 Map Register
0x446	UnivOutMap3	UINT16	Front Panel Universal Output 3 Map Register
0x448	UnivOutMap4	UINT16	Front Panel Universal Output 4 Map Register
0x44A	UnivOutMap5	UINT16	Front Panel Universal Output 5 Map Register
0x44C	UnivOutMap6	UINT16	Front Panel Universal Output 6 Map Register
0x44E	UnivOutMap7	UINT16	Front Panel Universal Output 7 Map Register
0x450	UnivOutMap8	UINT16	Front Panel Universal Output 8 Map Register
0x452	UnivOutMap9	UINT16	Front Panel Universal Output 9 Map Register
0x480	TBOutMap0	UINT16	Transition Board Output 0 Map Register
0x482	TBOutMap1	UINT16	Transition Board Output 1 Map Register
0x484	TBOutMap2	UINT16	Transition Board Output 2 Map Register
0x486	TBOutMap3	UINT16	Transition Board Output 3 Map Register
0x488	TBOutMap4	UINT16	Transition Board Output 4 Map Register
0x48A	TBOutMap5	UINT16	Transition Board Output 5 Map Register
0x48C	TBOutMap6	UINT16	Transition Board Output 6 Map Register
0x48E	TBOutMap7	UINT16	Transition Board Output 7 Map Register
0x490	TBOutMap8	UINT16	Transition Board Output 8 Map Register
0x492	TBOutMap9	UINT16	Transition Board Output 9 Map Register
0x494	TBOutMap10	UINT16	Transition Board Output 10 Map Register
0x496	TBOutMap11	UINT16	Transition Board Output 11 Map Register
0x498	TBOutMap12	UINT16	Transition Board Output 12 Map Register
0x49A	TBOutMap13	UINT16	Transition Board Output 13 Map Register
0x49C	TBOutMap14	UINT16	Transition Board Output 14 Map Register
0x49E	TBOutMap15	UINT16	Transition Board Output 15 Map Register
0x4A0	TBOutMap16	UINT16	Transition Board Output 16 Map Register
0x4A2	TBOutMap17	UINT16	Transition Board Output 17 Map Register
0x4A4	TBOutMap18	UINT16	Transition Board Output 18 Map Register
0x4A6	TBOutMap19	UINT16	Transition Board Output 19 Map Register
0x4A8	TBOutMap20	UINT16	Transition Board Output 20 Map Register
0x4AA	TBOutMap21	UINT16	Transition Board Output 21 Map Register
0x4AC	TBOutMap22	UINT16	Transition Board Output 22 Map Register
0x4AE	TBOutMap23	UINT16	Transition Board Output 23 Map Register
0x4B0	TBOutMap24	UINT16	Transition Board Output 24 Map Register
0x4B2	TBOutMap25	UINT16	Transition Board Output 25 Map Register
0x4B4	TBOutMap26	UINT16	Transition Board Output 26 Map Register
0x4B6	TBOutMap27	UINT16	Transition Board Output 27 Map Register

0x4B8	TBOutMap28	UINT16	Transition Board Output 28 Map Register
0x4BA	TBOutMap29	UINT16	Transition Board Output 29 Map Register
0x4BC	TBOutMap30	UINT16	Transition Board Output 30 Map Register
0x4BE	TBOutMap31	UINT16	Transition Board Output 31 Map Register
0x500	FPInMap0	UINT32	Front Panel Input 0 Mapping Register
0x504	FPInMap1	UINT32	Front Panel Input 1 Mapping Register
0x600	CML4Pat00	UINT32	20 bit output pattern for state low
0x604	CML4Pat01	UINT32	20 bit output pattern for state rising edge
0x608	CML4Pat10	UINT32	20 bit output pattern for state falling edge
0x60C	CML4Pat11	UINT32	20 bit output pattern for state high
0x610	CML4Ena	UINT32	CML 4 Output Control Register
0x614	CML4HP	UINT16	CML 4 Output High Period Count
0x616	CML4LP	UINT16	CML 4 Output Low Period Count
0x618	CML4Samp	UINT32	CML 4 Output Number of 20 bit word patterns
0x620	CML5Pat00	UINT32	20 bit output pattern for state low
0x624	CML5Pat01	UINT32	20 bit output pattern for state rising edge
0x628	CML5Pat10	UINT32	20 bit output pattern for state falling edge
0x62C	CML5Pat11	UINT32	20 bit output pattern for state high
0x630	CML5Ena	UINT32	CML 5 Output Control Register
0x634	CML5HP	UINT16	CML 5 Output High Period Count
0x636	CML5LP	UINT16	CML 5 Output Low Period Count
0x638	CML5Samp	UINT32	CML 5 Output Number of 20 bit word patterns
0x640	CML6Pat00	UINT32	20 bit output pattern for state low
0x644	CML6Pat01	UINT32	20 bit output pattern for state rising edge
0x648	CML6Pat10	UINT32	20 bit output pattern for state falling edge
0x64C	CML6Pat11	UINT32	20 bit output pattern for state high
0x650	CML6Ena	UINT32	CML 6 Output Control Register
0x654	CML6HP	UINT16	CML 6 Output High Period Count
0x656	CML6LP	UINT16	CML 6 Output Low Period Count
0x658	CML6Samp	UINT32	CML 6 Output Number of 20 bit word patterns
0x800 – 0xFFF	DataBuf		Data Buffer Receive Memory
0x1000 – 0x17FF			Diagnostics counters
0x1800 – 0x1FFF	TxDataBuf		Data Buffer Transmit Memory
0x2000 – 0x3FFF	EventLog		512 x 16 byte position Event Log
0x4000 – 0x5FFF	MapRam1		Event Mapping RAM 1
0x6000 – 0x7FFF	MapRam2		Event Mapping RAM 2
0x8000 – 0x80FF	configROM		
0x8100 –	scratchRAM		

0x81FF		
0x8200 – 0x82FF	SFPEEPROM	SFP Transceiver EEPROM contents (SFP address 0xA0)
0x8300 – 0x83FF	SFPDIAG	SFP Transceiver diagnostics (SFP address 0xA2)
0x20000 – 0x23FFF	CML4PMEM GTX0MEM	Pattern memory: 8k bytes CML output 4 (VME-EVR-230RF) 16k bytes GTX output 0 (cPCI-EVRTG-300)
0x24000 – 0x27FFF	CML5PMEM GTX1MEM	Pattern memory: 8k bytes CML output 5 (VME-EVR-230RF) 16k bytes GTX output 1 (cPCI-EVRTG-300)
0x28000 – 0x2BFFF	CML6PMEM GTX2MEM	Pattern memory: 8k bytes CML output 6 (VME-EVR-230RF) 16k bytes GTX output 2 (cPCI-EVRTG-300)
0x2C000 – 0x2FFFF	GTX3MEM	Pattern memory: 16k bytes GTX output 3 (cPCI-EVRTG-300)
0x30000 – 0x33FFF	GTX4MEM	Pattern memory: 16k bytes GTX output 4 (cPCI-EVRTG-300)
0x34000 – 0x37FFF	GTX5MEM	Pattern memory: 16k bytes GTX output 5 (cPCI-EVRTG-300)
0x38000 – 0x3BFFF	GTX6MEM	Pattern memory: 16k bytes GTX output 6 (cPCI-EVRTG-300)
0x3C000 – 0x3FFFF	GTX7MEM	Pattern memory: 16k bytes GTX output 7 (cPCI-EVRTG-300)

Status Register

address	bit 31	bit 30	bit 29	Bit 28	bit 27	bit 26	bit 25	bit 24
0x000	DBUS7	DBUS6	DBUS5	DBUS4	DBUS3	DBUS2	DBUS1	DBUS0

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x001								LEGVIO

address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x002								

address	bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x003	SFPMOD	LINK	FIFOSTP					

Bit	Function
DBUS7	Read status of DBUS bit 7
DBUS6	Read status of DBUS bit 6
DBUS5	Read status of DBUS bit 5
DBUS4	Read status of DBUS bit 4
DBUS3	Read status of DBUS bit 3

DBUS2	Read status of DBUS bit 2
DBUS1	Read status of DBUS bit 1
DBUS0	Read status of DBUS bit 0
LEGVIO	Legacy VIO (series 100, 200 and 230)
SFPMOD	SFP module status: '0' – plugged in '1' – no module installed
LINK	Link status: '0' – link down '1' – link up
FIFOSTP	Event FIFO stopped flag

Control Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	Obit 26	bit 25	bit 24
0x004	EVREN	EVFWD	TXLP	RXLP				

address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x006		TSDBUS	RSTS			LTS	MAPEN	MAPRS

address	bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x007	LOGRS	LOGEN	LOGDIS	LOGSE	RSFIFO			

Bit	Function
EVREN	Event Receiver Master enable
TXLP	Transmitter loopback: 0 – Receive signal from SFP transceiver (normal operation) 1 – Loopback EVR TX into EVR RX
RXLP	Receiver loopback: 0 – Transmit signal from EVR on SFP transceiver TX 1 – Loopback SFP RX on SFP TX
TSDBUS	Use timestamp counter clock on DBUS4
RSTS	Reset Timestamp. Write 1 to reset timestamp event counter and timestamp latch.
LTS	Latch Timestamp: Write 1 to latch timestamp from timestamp event counter to timestamp latch.
MAPEN	Event mapping RAM enable.
MAPRS	Mapping RAM select bit for event decoding: 0 – select mapping RAM 1 1 – select mapping RAM 2.
LOGRS	Reset Event Log. Write 1 to reset log.
LOGEN	Enable Event Log. Write 1 to (re)enable event log.
LOGDIS	Disable Event Log. Write 1 to disable event log.
LOGSE	Log Stop Event Enable.
RSFIFO	Reset Event FIFO. Write 1 to clear event FIFO.

Interrupt Flag Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x008								

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x00b		IFLINK	IFDBUF	IFHW	IFEV	IFHB	IFFF	IFVIO

Bit	Function
IFLINK	Link state change interrupt flag
IFDBUF	Data buffer flag
IFHW	Hardware interrupt flag (mapped signal)
IFEV	Event interrupt flag
IFHB	Heartbeat interrupt flag
IFFF	Event FIFO full flag
IFVIO	Receiver violation flag

Interrupt Enable Register

address	Bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x00c	IRQEN							

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x00f		IELINK	IEDBUF	IEHW	IEEV	IEHB	IEFF	IEVIO

Bit	Function
IRQEN	Master interrupt enable: 0 – disable all interrupts 1 – allow interrupts
IELINK	Link state change interrupt flag
IEDBUF	Data buffer interrupt enable
IEHW	Hardware interrupt enable (mapped signal)
IEEV	Event interrupt enable
IEHB	Heartbeat interrupt enable
IEFF	Event FIFO full interrupt enable
IEVIO	Receiver violation interrupt enable

Hardware Interrupt Mapping Register

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x013	Mapping ID (see Table 1 for mapping IDs)							

Receive Data Buffer Control and Status Register

address	Bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x022	DBRX/ DBENA	DBRDY/ DBDIS	DBCS	DBEN	RXSIZE(11:8)			

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

0x023	RXSIZE(7:0)

Bit	Function
DBRX	Data Buffer Receiving (read-only)
DBENA	Set-up for Single Reception (write '1' to set-up)
DBRDY	Data Buffer Transmit Complete / Interrupt Flag
DBDIS	Stop Reception (write '1' to stop/disable)
DBCS	Data Buffer Checksum Error (read-only) Flag is cleared by writing '1' to DBRX or DBRDY or disabling data buffer
DBEN	Data Buffer Enable Data Buffer Mode '0' – Distributed bus not shared with data transmission, full speed distributed bus '1' – Distributed bus shared with data transmission, half speed distributed bus
RXSIZE	Data Buffer Received Buffer Size (read-only)

Transmit Data Buffer Control Register

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x025				TXCPT	TXRUN	TRIG	ENA	MODE

address	bit 15	Bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x026						DTSZ(10:8)		

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x027	DTSZ(7:2)						0	0

Bits	Function
TXCPT	Data Buffer Transmission Complete
TXRUN	Data Buffer Transmission Running – set when data transmission has been triggered and has not been completed yet
TRIG	Data Buffer Trigger Transmission Write '1' to start transmission of data in buffer
ENA	Data Buffer Transmission enable '0' – data transmission engine disabled '1' – data transmission engine enabled
MODE	Distributed bus sharing mode '0' – distributed bus not shared with data transmission '1' – distributed bus shared with data transmission
DTSZ(10:8)	Data Transfer size 4 bytes to 2k in four byte increments

FPGA Firmware Version Register

address	bit 31	bit 27	bit 26	bit 24
0x02C	EVR = 0x1		Form Factor	

address	bit 23	bit 8
0x02D	Reserved	

address	bit 7							bit 0
0x02F	Version ID							

Bits	Function
Form Factor	0 – CompactPCI 3U
	1 – PMC
	2 – VME64x
	3 – CompactRIO
	4 – CompactPCI 6U

Event Counter Clock Prescaler Register

address	bit 15							bit 0
0x042	Timestamp Event Counter Clock Prescaler Register							

Microsecond Divider Register

address	bit 15							bit 0
0x04e	Rounded integer value of 1 µs * event clock							

For 100 MHz event clock this register should read 100, for 50 MHz event clock this register should read 50. This value is used e.g. for the heartbeat timeout.

Clock Control Register

address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x052	RECDCM RUN	RECDCM INITDONE	RECDCM PSDONE	EVDCM STOPPED	EVDCM LOCKED	EVDCM PSDONE	CGLOCK	RECDCM PSDEC

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x053	RECDCM PSINC	RECDCM RES	EVDCM PSDEC	EVDCM PSINC	EVDCM SRUN	EVDCM SRES	EVDCM RES	EVCLKSEL

Bit	Function
CGLOCK	Micrel SY87739L locked (read-only)
Other bits	n/a on cPCI-EVR

Seconds Shift Register

address	bit 31							bit 0
0x05c	Seconds Shift Register (read-only)							

Seconds Counter Register

address	bit 31							bit 0
0x060	Seconds Counter Register (read-only)							

Timestamp Event Counter Register

address	bit 31							bit 0

0x064	Timestamp Event Counter Register (read-only)

Seconds Latch Register

address	bit 31	bit 0
0x068	Seconds Latch Register (read-only)	

Timestamp Event Latch Register

address	bit 31	bit 0
0x06c	Timestamp Event Latch Register (read-only)	

FIFO Seconds Register

address	bit 31	bit 0
0x070	FIFO Seconds Register (read-only)	

FIFO Timestamp Register

address	bit 31	bit 0
0x074	FIFO Timestamp Register (read-only)	

FIFO Event Register

address	bit 7	bit 0
0x07b	FIFO Event Code Register (read-only)	

Note that reading the FIFO event code registers pulls the event code and timestamp/seconds value from the FIFO for access. The correct order to read an event from FIFO is to first read the event code register and after this the timestamp/seconds registers in any order. Every read access to the FIFO event register pulls a new event from the FIFO if it is not empty.

Event Log Status Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x07C	LOGOV							

address	bit 15	bit 9	bit 8	bit 0
0x07E			Log writing pointer	

SY87739L Fractional Divider Configuration Word

address	bit 31	bit 0
0x080	SY87739L Fractional Divider Configuration Word	

Configuration Word	Frequency with 24 MHz reference oscillator
0x00DE816D	125 MHz
0x00FE816D	124.95 MHz
0x0C928166	124.908 MHz
0x018741AD	119 MHz
0x072F01AD	114.24 MHz

0x049E81AD	106.25 MHz
0x008201AD	100 MHz
0x025B41ED	99.956 MHz
0x0187422D	89.25 MHz
0x0082822D	81 MHz
0x0106822D	80 MHz
0x019E822D	78.900 MHz
0x018742AD	71.4 MHz
0x0C9282A6	62.454 MHz
0x009743AD	50 MHz
0x025B43AD	49.978 MHz
0x0176C36D	49.965 MHz

Prescaler 0 Register

address	Bit 15	bit 0
0x102	Prescaler 0 Register	

Prescaler 1 Register

address	Bit 15	bit 0
0x106	Prescaler 1 Register	

Prescaler 2 Register

address	Bit 15	bit 0
0x10a	Prescaler 2 Register	

Pulse Generator Registers

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x203	PxOUT	PxSWS	PxSWR	PxPOL	PxMRE	PxMSE	PxMTE	PxENA

address	bit 31	bit 0
0x204	Pulse Generator Prescaler Register	

address	bit 31	bit 0
0x208	Pulse Generator Delay Register	

address	bit 31	bit 0
0x20C	Pulse Generator Width Register	

Note: addresses shown above are for pulse generator 0.

bit	Function
PxOUT	Pulse Generator Output (read-only)
PxSWS	Pulse Generator Software Set
PxSWC	Pulse Generator Software Reset
PxPOL	Pulse Generator Output Polarity 0 – normal polarity 1 – inverted polarity

PxMRE	Pulse Generator Event Mapping RAM Reset Event Enable 0 – Reset events disabled 1 – Mapped Reset Events reset pulse generator output
PxMSE	Pulse Generator Event Mapping RAM Set Event Enable 0 – Set events disabled 1 – Mapped Set Events set pulse generator output
PxMTE	Pulse Generator Event Mapping RAM Trigger Event Enable 0 – Event Triggers disabled 1 – Mapped Trigger Events trigger pulse generator
PxENA	Pulse Generator Enable 0 – generator disabled 1 – generator enabled

Front Panel Output Mapping Registers

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x401	Front panel OUT0 Mapping ID (see Table 1 for mapping IDs)							
0x403	Front panel OUT1 Mapping ID							
0x405	Front panel OUT2 Mapping ID							
0x407	Front panel OUT3 Mapping ID							
0x409	Front panel OUT4 Mapping ID							
0x40B	Front panel OUT5 Mapping ID							
0x40D	Front panel OUT6 Mapping ID							
0x40F	Front panel OUT7 Mapping ID							

Notes:
cPCI-EVR does not have any Front panel outputs.
PMC-EVR has three front panel outputs OUT0 to OUT2.
VME-EVR-230 has eight Front panel outputs OUT0 to OUT7.
VME-EVR-230RF has seven Front panel outputs OUT0 to OUT3 (TTL level), OUT4 to OUT6 CML level (see section about CML outputs for details).

Universal I/O Output Mapping Registers

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x441	Universal I/O UNIV0 Mapping ID (see Table 1 for mapping IDs)							
0x443	Universal I/O UNIV1 Mapping ID							
0x445	Universal I/O UNIV2 Mapping ID							
0x447	Universal I/O UNIV3 Mapping ID							
0x449	Universal I/O UNIV4 Mapping ID (optional cPCI side-by-side module)							
0x44b	Universal I/O UNIV5 Mapping ID (optional cPCI side-by-side module)							
0x44d	Universal I/O UNIV6 Mapping ID (optional cPCI side-by-side module)							
0x44f	Universal I/O UNIV7 Mapping ID (optional cPCI side-by-side module)							
0x451	Universal I/O UNIV8 Mapping ID (optional cPCI side-by-side module)							
0x453	Universal I/O UNIV9 Mapping ID (optional cPCI side-by-side module)							

Notes:
cPCI-EVR has two Universal I/O slots (four outputs UNIV0 to UNIV3). An optional side-by-side module provides three more slots (six additional outputs UNIV4 to UNIV9).
PMC-EVR does not have any Universal I/O slots.
VME-EVR has two Universal I/O slots (four outputs UNIV0 to UNIV3).

Transition Board Output Mapping Registers

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x481	Transition Board Output TBOUT0 Mapping ID (see Table 1 for mapping IDs)							
0x483	Transition Board Output TBOUT1 Mapping ID							
0x485	Transition Board Output TBOUT2 Mapping ID							
…	…							

Notes:
cPCI-EVR does not have any Transition board outputs.

Front Panel Input Mapping Registers

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x500			EXTLV0	BCKLE0	EXTLE0	EXTED0	BCKEV0	EXTEV0

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x501	T0DB7	T0DB6	T0DB5	T0DB4	T0DB3	T0DB2	T0DB1	T0DB0

address	bit 15	bit 8
0x502	Backward Event Code Register for front panel input 0	

address	bit 7	bit 0
0x503	External Event Code Register for front panel input 0	

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x504			EXTLV1	BCKLE1	EXTLE1	EXTED1	BCKEV1	EXTEV1

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x505	T1DB7	T1DB6	T1DB5	T1DB4	T1DB3	T1DB2	T1DB1	T1DB0

address	bit 15	bit 8
0x506	Backward Event Code Register for front panel input 1	

address	bit 7	bit 0
0x507	External Event Code Register for front panel input 1	

bit	Function
EXTLVx	Backward HW Event Level Sensitivity for input x 0 – active high 1 – active low
BCKLEx	Backward HW Event Level Trigger enable for input x 0 – disable level events 1 – enable level events, send out backward event code every 1 us when input is active (see EXTLVx for level sensitivity)
EXTLEx	External HW Event Level Trigger enable for input x 0 – disable level events 1 – enable level events, apply external event code to active mapping RAM every 1 us when input is active (see EXTLVx for level sensitivity)
EXTEDx	Backward HW Event Edge Sensitivity for input x

0 – trigger on rising edge
1 – trigger on falling edge

BCKEVx Backward HW Event Edge Trigger Enable for input x
0 – disable backward HW event
1 – enable backward HW event, send out backward event code on detected edge of hardware input (see EXTEDx bit for edge)

EXTEVx External HW Event Enable for input x
0 – disable external HW event
1 – enable external HW event, apply external event code to active mapping RAM on edge of hardware input

TxDB7-TxDB0 Backward distributed bus bit enable:
0 – disable distributed bus bit
1 – enable distributed bus bit control from hardware input: e.g. when TxDB7 is '1' the hardware input x state is sent out on distributed bus bit 7.

CML Output Pattern Registers (CMLxPatxx)

bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
				19 MSB	18	17	16

bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
15	14	13	12	11	10	9	8

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
7	6	5	4	3	2	1	0 LSB

Bit 19 MSB is sent out first, LSB last
Note that GTX pattern registers are accessed through the first four address locations of the GTX pattern memory.

CML/GTX Output Control Register

Address	bit 31	bit 16
	Frequency mode trigger position	

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
	CMLRC	CMLTL	CMLMD			CMLRES	CMLPWD	CMLENA

CMLRC CML Pattern recycle
CMLTL CML Frequency mode trigger level
CMLMD CML Mode Select:
00 = classic mode
01 = frequency mode
10 = pattern mode
11 = undefined
CMLRES CML Reset
1 = reset CML output (default on EVR power up)

	0 = normal operation
CMLPWD	CML Power Down 1 = CML outputs powered down (default on EVR power up) 0 = normal operation
CMLENA	CML Enable 0 = CML output disabled (default on EVR power up) 1 = CML output enabled

SFP Module EEPROM and Diagnostics

Small Form Factor Pluggable (SFP) transceiver modules provide a means to identify the module by accessing an EEPROM. As an advanced feature some modules also support reading dynamic information including module temperature, receive and transmit power levels etc. from the module. The EVR gives access to all of this information through a memory window of 2 × 256 bytes. The first 256 bytes consist of the EEPROM values and the rest of the advanced values.

Byte # Decimal	**Field size (bytes)**	**Notes**	**Value Hex**
		BASE ID FIELDS	
0	1	Type of serial transceiver	03 = SFP transceiver
1	1	Extended identifier of type serial transceiver	04 = serial ID module definition
2	1	Code for connector type	07 = LC
3 – 10	8	Code for electronic compatibility or optical compatibility	
11	1	Code for serial encoding algorithm	
12	1	Nominal bit rate, units of 100 MBits/sec	
13	1	Reserved	
14	1	Link length supported for 9/125 µm fiber, units of km	
15	1	Link length supported for 9/125 µm fiber, units of 100 m	
16	1	Link length supported for 50/125 µm fiber, units of 10 m	
17	1	Link length supported for 62.5/125 µm fiber, units of 10 m	
18	1	Link length supported for copper, units of meters	
19	1	Reserved	
20 – 35	16	SFP transceiver vendor name (ASCII)	
36	1	Reserved	
37 – 39	3	SFP transceiver vendor IEEE company ID	
40 – 55	16	Part number provided by SFP transceiver vendor (ASCII)	
56 – 59	4	Revision level for part number	

		provided by vendor (ASCII)	
60 – 62	3	Reserved	
63	1	Check code for Base ID Fields	
		EXTENDED ID FIELDS	
64 – 65	2	Indicated which optional SFP signals are implemented	
66	1	Upper bit rate margin, units of %	
67	1	Lower bit rate margin, units of %	
68 – 83	16	Serial number provided by vendor (ASCII)	
84 – 91	8	Vendor's manufacturing date code	
92 – 94	3	Reserved	
95	1	Check code for the Extended ID Fields	
		VENDOR SPECIFIC ID FIELDS	
96 – 127	32	Vendor specific data	
128 – 255		Reserved	
		ENHANCED FEATURE SET MEMORY	
256 – 257	2	Temp H Alarm	Signed twos complement integer in increments of 1/256 °C
258 – 259	2	Temp L Alarm	Signed twos complement integer in increments of 1/256 °C
260 – 261	2	Temp H Warning	Signed twos complement integer in increments of 1/256 °C
262 – 263	2	Temp L Warning	Signed twos complement integer in increments of 1/256 °C
264 – 265	2	VCC H Alarm	Supply voltage decoded as unsigned integer in increments of 100 µV
266 – 267	2	VCC L Alarm	Supply voltage decoded as unsigned integer in increments of 100 µV
268 – 269	2	VCC H Warning	Supply voltage decoded as unsigned integer in increments of 100 µV
270 – 271	2	VCC L Warning	Supply voltage decoded as unsigned integer in increments of 100 µV
272 – 273	2	Tx Bias H Alarm	Laser bias current decoded as unsigned integer in increment of 2 µA
274 – 275	2	Tx Bias L Alarm	Laser bias current decoded as unsigned integer in increment of 2 µA
276 – 277	2	Tx Bias H Warning	Laser bias current decoded as

			unsigned integer in increment of 2 µA
278 – 279	2	Tx Bias L Warning	Laser bias current decoded as unsigned integer in increment of 2 µA
280 – 281	2	Tx Power H Alarm	Transmitter average optical power decoded as unsigned integer in increments of 0.1 µW
282 – 283	2	Tx Power L Alarm	Transmitter average optical power decoded as unsigned integer in increments of 0.1 µW
284 – 285	2	Tx Power H Warning	Transmitter average optical power decoded as unsigned integer in increments of 0.1 µW
286 – 287	2	Tx Power L Warning	Transmitter average optical power decoded as unsigned integer in increments of 0.1 µW
288 – 289	2	Rx Power H Alarm	Receiver average optical power decoded as unsigned integer in increments of 0.1 µW
290 – 291	2	Rx Power L Alarm	Receiver average optical power decoded as unsigned integer in increments of 0.1 µW
292 – 293	2	Rx Power H Warning	Receiver average optical power decoded as unsigned integer in increments of 0.1 µW
294 – 295	2	Rx Power L Warning	Receiver average optical power decoded as unsigned integer in increments of 0.1 µW
296 – 311	16	Reserved	
312 – 350		External Calibration Constants	
351	1	Checksum for Bytes 256 – 350	
352 – 353	2	Real Time Temperature	Signed twos complement integer in increments of 1/256 °C
354 – 355	2	Real Time VCC Power SupplyVoltage	Supply voltage decoded as unsigned integer in increments of 100 µV
356 – 357	2	Real Time Tx Bias Current	Laser bias current decoded as unsigned integer in increment of 2 µA

358 – 359	2	Real Time Tx Power	Transmitter average optical power decoded as unsigned integer in increments of 0.1 µW
360 – 361	2	Real Time Rx Power	Receiver average optical power decoded as unsigned integer in increments of 0.1 µW
362 – 365	4	Reserved	
366	1	Status/Control	bit 7: TX_DISABLE State bit 6 – 3: Reserved bit 2: TX_FAULT State bit 1: RX_LOS State bit 0: Data Ready (Bar)
367	1	Reserved	
368	1	Alarm Flags	bit 7: Temp High Alarm bit 6: Temp Low Alarm bit 5: VCC High Alarm bit 4: VCC Low Alarm bit 3: Tx Bias High Alarm bit 2: Tx Bias Low Alarm bit 1: Tx Power High Alarm bit 0: Tx Power Low Alarm
369	1	Alarm Flags cont.	bit 7: Rx Power High Alarm bit 6: Rx Power Low Alarm bit 5 – 0: Reserved
370 – 371	2	Reserved	
372	1	Warning Flags	bit 7: Temp High Warning bit 6: Temp Low Warning bit 5: VCC High Warning bit 4: VCC Low Warning bit 3: Tx Bias High Warning bit 2: Tx Bias Low Warning bit 1: Tx Power High Warning bit 0: Tx Power Low Warning
373	1	Warning Flags cont.	bit 7: Rx Power High Warning bit 6: Rx Power Low Warning bit 5 – 0: Reserved
374 – 511		Reserved/Vendor Specific	

Application Programming Interface (API)

A Linux device driver and application interface is provided to setup up the Event Receiver.

Function Reference

int EvrOpen(struct MrfErRegs **pEr, char *device_name);

Description		Opens the EVR device for access. Simultaneous accesses are allowed.
Parameters	struct MrfErRegs **pEr	EvgOpen returns pointer to EVR registers by memory mapping the I/O registers into user space.
	char *device_name	Holds the device name of the EVR, e.g. /dev/ega3. The device names are set up by the module_load script of the device driver.
Return value		Return file descriptor on success. Returns -1 on error.

int EvrClose(int fd);

Description		Closes the EVR device after opening by EvrOpen.
Parameters	int fd	File descriptor returned by EvrOpen
Return value		Returns zero on success. Returns -1 on error.

int EvrEnable(volatile struct MrfErRegs *pEr, int state);

Description		Enables the EVR and allows reception of events.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int state	0: disable 1: enable
Return value		Returns zero when EVR disabled Returns non-zero when EVR enabled

int EvrGetEnable(volatile struct MrfErRegs *pEr);

Description		Retrieves state of the EVR.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Returns zero when EVR disabled Returns non-zero when EVR enabled

void EvrDumpStatus(volatile struct MrfErRegs *pEr);

Description		Dump EVR status.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		None

int EvrGetViolation(volatile struct MrfErRegs *pEr, int clear);

Description		Get/clear EVR link violation status.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int clear	0: don't clear 1: clear status
Return value		Returns 0 when no violation detected. Return non-zero when violation detected.

void EvrDumpMapRam(volatile struct MrfErRegs *pEr, int ram);

Description		Dump EVR mapping RAM.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ram	Number of RAM: 0 or 1
Return value		None

int EvrMapRamEnable(volatile struct MrfErRegs *pEr, int ram, int enable);

Description		Enable/disable EVR mapping RAM.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ram	Number of RAM: 0 or 1
	int enable	0: disable RAM 1: enable RAM
Return value		None

int EvrSetForwardEvent(volatile struct MrfErRegs *pEr, int ram, int code, int enable);

Description		Enable/disable EVR event forwarding.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ram	Number of mapping RAM: 0 or 1
	int code	Event code to enable/disable event forwarding
	int enable	0: disable event forwarding for code 1: enable event forwarding for code
Return value		None

int EvrEnableEventForwarding(volatile struct MrfErRegs *pEr, int state);

Description		Enables forwarding of enabled event codes.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.

	int state	0: disable forwarding 1: enable forwarding
Return value		Returns zero when forwarding disabled Returns non-zero when forwarding enabled

int EvrGetEventForwarding(volatile struct MrfErRegs *pEr);

Description		Retrieves state of event forwarding.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Returns zero when forwarding disabled Returns non-zero when forwarding enabled

int EvrSetLedEvent(volatile struct MrfErRegs *pEr, int ram, int code, int enable);

Description		Enable/disable EVR led event (Front panel led will flash up for enabled event codes).
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ram	Number of mapping RAM: 0 or 1
	int code	Event code to enable/disable led event for
	int enable	0: disable led event for code 1: enable led event for code
Return value		None

int EvrSetFIFOEvent(volatile struct MrfErRegs *pEr, int ram, int code, int enable);

Description		Enable/disable storing specified event code into FIFO.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ram	Number of mapping RAM: 0 or 1
	int code	Event code to enable/disable
	int enable	0: disable storing event code in FIFO 1: enable storing event code in FIFO
Return value		None

int EvrSetLatchEvent(volatile struct MrfErRegs *pEr, int ram, int code, int enable);

Description		Enable/disable latching timestamp on specified event code.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ram	Number of mapping RAM: 0 or 1
	int code	Event code to enable/disable
	int enable	0: disable latching of timestamp on event code 1: enable latching of timestamp upon

		reception of event code
Return value		None

int EvrSetLogStopEvent(volatile struct MrfErRegs *pEr, int ram, int code, int enable);

Description		Enable/disable stopping of writes to event log on reception of event code.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ram	Number of mapping RAM: 0 or 1
	int code	Event code to enable/disable
	int enable	0: disable stop log event 1: stop log writes upon reception of event code
Return value		None

int EvrClearFIFO(volatile struct MrfErRegs *pEr);

Description		Clear EVR Event FIFO.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		None.

int EvrGetFIFOEvent(volatile struct MrfErRegs *pEr, struct FIFOEvent *fe);

Description		Get one Event from EVR Event FIFO.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	struct FIFOEvent *fe	Pointer to structure to place event in.

```
struct FIFOEvent {
  u32 TimestampHigh;
  u32 TimestampLow;
  u32 EventCode;
};
```

Return value		0 – Event retrieved successfully -1 – Event FIFO was empty

int EvrEnableLogStopEvent(volatile struct MrfErRegs *pEr, int enable);

Description		Enable/disable stopping of writing to event log on reception of event codes with STOP Log mapping bit set.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int enable	0: disable stop log event 1: stop log writes upon reception of event codes with STOP log mapping bit set.
Return value		Returns zero when stop events disabled

		Returns non-zero when stop events enabled

int EvrGetLogStopEvent(volatile struct MrfErRegs *pEr);

Description		Check if log stop events are enabled.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Returns zero when stop events disabled Returns non-zero when stop events enabled

int EvrEnableLog(volatile struct MrfErRegs *pEr, int enable);

Description		Enable/disable writing to log.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int enable	0: disable writes to log 1: enable writes to log
Return value		Returns zero when log enabled Returns non-zero when log stopped.

int EvrGetLogState(volatile struct MrfErRegs *pEr, int enable);

Description		Get log state.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Returns zero when logging enabled Returns non-zero when logging stopped.

int EvrGetLogStart(volatile struct MrfErRegs *pEr);

Description		Get log start position.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Returns relative address to first log entry in log ring buffer.

int EvrGetLogEntries(volatile struct MrfErRegs *pEr);

Description		Get number of entries in log.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Returns number of entries in log (0 to 512).

void EvrDumpFIFO(volatile struct MrfErRegs *pEr);

Description		Dump EVR FIFO on stdout.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		None

int EvrClearLog(volatile struct MrfErRegs *pEr);

Description		Empty EVR Event Log.

Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		None.

void EvrDumpLog(volatile struct MrfErRegs *pEr);

Description		Print out full EVR event log on stdout.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		None

int EvrSetPulseMap(volatile struct MrfErRegs *pEr, int ram, int code, int trig, int set, int clear);

Description		Set up pulse generators for event codes.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ram	Number of mapping RAM: 0 or 1
	int code	Event code affected
	int trig	0: no change 1: Trigger pulse generator from event code
	int set	0: no change 1: Set pulse high with this event code
	int clear	0: no change 1: Pull pulse low with this event code
Return value		None

int EvrClearPulseMap(volatile struct MrfErRegs *pEr, int ram, int code, int trig, int set, int clear);

Description		Set up pulse generators for event codes.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ram	Number of mapping RAM: 0 or 1
	int code	Event code affected
	int trig	0: no change 1: Don’t trigger pulse generator from this event code
	int set	0: no change 1: Don’t set pulse high with this event code
	int clear	0: no change 1: Don’t pull pulse low with this event code
Return value		None

int EvrSetPulseParams(volatile struct MrfErRegs *pEr, int pulse, int presc, int delay, int width);

Description		Set pulse generator parameters.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.

	int pulse	Number of pulse generator 0-9
	int presc	Prescaler value
	int delay	Delay Value
	int width	Width Value
Return value		Returns 0 on success, -1 on error

void EvrDumpPulses(volatile struct MrfErRegs *pEr, int pulses);

Description		Dump EVR pulse generator settings.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int pulses	Number of pulse generators to dump
Return value		None

int EvrSetPulseProperties(volatile struct MrfErRegs *pEr, int pulse, int polarity, int map_reset_ena, int map_set_ena, int map_trigger_ena, int enable);

Description		Set pulse generator properties.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int pulse	Number of pulse generator 0-9
	int polarity	0: normal polarity 1: inverted polarity
	int map_reset_ena	0: disable reset input 1: enable reset input
	int map_set_ena	0: disable set input 1: enable set input
	int map_trigger_ena	0: disable trigger input 1: enable trigger input
	int enable	0: pulse output disabled 1: pulse output enabled
Return value		Returns 0 on success, -1 on error

int EvrSetUnivOutMap(volatile struct MrfErRegs *pEr, int output, int map);

Description		Set up universal output mappings.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int output	Universal Output number
	int map	Signal mapping (see erapi.h for details)
Return value		Returns 0 on success, -1 on error

void EvrDumpUnivOutMap(volatile struct MrfErRegs *pEr, int outputs);

Description		Dump EVR Universal output mappings.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int outputs	Number of outputs to dump
Return value		None

int EvrSetFPOutMap(volatile struct MrfErRegs *pEr, int output, int map);

Description		Set up front panel output mappings.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int output	Front Panel Output number
	int map	Signal mapping (see erapi.h for details)
Return value		Returns 0 on success, -1 on error

void EvrDumpFPOutMap(volatile struct MrfErRegs *pEr, int outputs);

Description		Dump EVR Front panel output mappings.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int outputs	Number of outputs to dump
Return value		None

int EvrSetTBOutMap(volatile struct MrfErRegs *pEr, int output, int map);

Description		Set up Transition board output mappings.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int output	Transition Board Output number
	int map	Signal mapping (see erapi.h for details)
Return value		Returns 0 on success, -1 on error

void EvrDumpTBOutMap(volatile struct MrfErRegs *pEr, int outputs);

Description		Dump EVR Transition board output mappings.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int outputs	Number of outputs to dump
Return value		None

void EvrIrqAssignHandler(volatile struct MrfErRegs *pEr, int fd, void (*handler)(int));

Description		Assign EVR interrupt handler.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int fd	File descriptor returned by EvrOpen
	void (*handler)(int)	Pointer to interrupt handler function
Return value		None

int EvrIrqEnable(volatile struct MrfErRegs *pEr, int mask);

Description		Enable EVR interrupts.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int mask	Interrupt mask (see erapi.h) for mask bits.
Return value		Returns mask read back from EVR.

int EvrGetIrqFlags(volatile struct MrfErRegs *pEr);

Description		Get EVR interrupt flags.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Returns EVR interrupt flags.

int EvrClearIrqFlags(volatile struct MrfErRegs *pEr, int mask);

Description		Clears EVR interrupt flags.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int mask	Interrupt clear mask (see erapi.h) for flag bits.
Return value		Returns flags read back from EVR.

void EvrIrqHandled(int fd);

Description		Function to call at the end of interrupt handler function.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int fd	File descriptor returned by EvrOpen
Return value		None

int EvrSetPulseIrqMap(volatile struct MrfErRegs *pEr, int map);

Description		Set up interrupt mappings.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register

		base.
	int map	Signal mapping (see erapi.h for details)
Return value		Returns 0 on success, -1 on error

int EvrUnivDlyEnable(volatile struct MrfErRegs *pEr, int dlymod, int enable);

Description		Enable/disable UNIV-LVPECL-DLY output.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int dlymod	Number of UNIV-LVPECL-DLY module: 0 – module in slot #1 (UNIV0/1) 1 – module in slot #2 (UNIV2/3)
	int enable	0 – disable module output 1 – enable module output
Return value		Returns 0 on success, -1 on error

int EvrUnivDlySetDelay(volatile struct MrfErRegs *pEr, int dlymod, int dly0, int dly1);

Description		Enable/disable UNIV-LVPECL-DLY output.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int dlymod	Number of UNIV-LVPECL-DLY module: 0 – module in slot #1 (UNIV0/1) 1 – module in slot #2 (UNIV2/3)
	int dly0	Delay value for output 0/2: 0 – shortest delay 1023 – longest delay (approx. 9-10 ps/step)
	int dly01	Delay value for output 1/3: 0 – shortest delay 1023 – longest delay (approx. 9-10 ps/step)
Return value		Returns 0 on success, -1 on error

int EvrSetFracDiv(volatile struct MrfErRegs *pEr, int fracdiv);

Description		Set fractional divider control word which provides reference frequency for receiver.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int fracdiv	Fractional divider control word
Return value		Returns control word written

int EvrGetFracDiv(volatile struct MrfErRegs *pEr);

Description		Get fractional divider control word which provides reference frequency for receiver.

Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Returns control word

int EvrSetDBufMode(volatile struct MrfErRegs *pEr, int enable);

Description		Enable/disable data buffer mode. When data buffer mode is enabled every other distributed bus byte is reserved for data transmission thus the distributed bus bandwidth is halved.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int enable	0 – disable data buffer mode 1 – enable data buffer mode
Return value		Data buffer status (see Receive Data Buffer Control and Status Register on page 34 for bit definitions).

int EvrGetDBufStatus(volatile struct MrfErRegs *pEr);

Description		Get data buffer mode. When data buffer mode is enabled every other distributed bus byte is reserved for data transmission thus the distributed bus bandwidth is halved.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Data buffer status (see Receive Data Buffer Control and Status Register on page 34 for bit definitions).

int EvrReceiveDBuf(volatile struct MrfErRegs *pEr, int enable);

Description		Enable reception of data buffer. After reception of a data buffer further reception is disabled until re-enabled by software.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int enable	0 – disable data buffer reception. 1 – enable data buffer reception
Return value		Data buffer status (see Receive Data Buffer Control and Status Register on page 34 for definitions).

int EvrGetDBuf(volatile struct MrfErRegs *pEr, char *dbuf, int size);

Description		Receive data buffer data.

Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	char *dbuf	Pointer to local data buffer
	int size	Size of dbuf buffer.
Return value		Size of received buffer. -1 on error (no buffer received, local buffer too small or checksum error)

int EvrSetTimestampDivider(volatile struct MrfErRegs *pEr, int div);

Description		Set timestamp counter divider
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int div	Timestamp divider value: 0 – count timestamp events (or use DBUS4 as clock) 1 to 65535 – count at event clock/value rate
Return value		Return divider value.

int EvrSetTimestampDBus(volatile struct MrfErRegs *pEr, int enable);

Description		Control timestamp counter count from distributed bus bit 4 (DBUS4).
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int enable	0 – disable counting from DBUS4 1 – enable timestamp counting from DBUS4. Note: Timestamp counter has to be 0.
Return value		

int EvrGetTimestampCounter(volatile struct MrfErRegs *pEr);

Description		Get Timestamp Counter value
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Timestamp Counter value

int EvrGetSecondsCounter(volatile struct MrfErRegs *pEr);

Description		Get Timestamp Seconds Counter value
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Timestamp Seconds Counter value

int EvrGetTimestampLatch(volatile struct MrfErRegs *pEr);

Description		Get Timestamp Latch value

Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Timestamp Latch value

int EvrGetSecondsLatch(volatile struct MrfErRegs *pEr);

Description		Get Timestamp Seconds Latch value
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Timestamp Seconds Latch value

int EvrSetPrescaler(volatile struct MrfErRegs *pEr, int presc, int div);

Description		Set prescaler divider
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int presc	Number of prescaler
	int div	Prescaler divider value: 1 to 65535 – count at event clock/value rate
Return value		Return divider value.

int EvrSetExtEvent(volatile struct MrfErRegs *pEr, int ttlin, int code, int edge_enable, int level_enable);

Description		Set external event code
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ttlin	Number of front panel input: 0, 1
	int code	Event code to generate on detected edge/level
	int edge_enable	0 – disable 1 – enable events on active edge
	int level_enable	0 – disable 1 – enable sending out event every 1 us on active level
Return value		0 – successful -1 – error

int EvrSetBackEvent(volatile struct MrfErRegs *pEr, int ttlin, int code, int edge_enable, int level_enable);

Description		Set backwards event code
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ttlin	Number of front panel input: 0, 1
	int code	Event code to send out on detected edge/level
	int edge_enable	0 – disable 1 – enable events on active edge

	int level_enable	0 – disable 1 – enable sending out event every 1 us on active level
Return value		0 – successful -1 – error

int EvrSetExtEdgeSensitivity(volatile struct MrfErRegs *pEr, int ttlin, int edge);

Description		Set external input edge sensitivity
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ttlin	Number of front panel input: 0, 1
	int edge	0 – detect rising edges 1 – detect falling edges
Return value		0 – successful -1 – error

int EvrSetExtLevelSensitivity(volatile struct MrfErRegs *pEr, int ttlin, int level);

Description		Set external input edge sensitivity
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int ttlin	Number of front panel input: 0, 1
	int level	0 – detect high level (active high) 1 – detect low level (active low)
Return value		0 – successful -1 – error

int EvrSetTxDBufMode(volatile struct MrfErRegs *pEr, int enable);

Description		Enable/disable transmitter data buffer mode. When data buffer mode is enabled every other distributed bus byte is reserved for data transmission thus the distributed bus bandwidth is halved.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	int enable	0 – disable transmitter data buffer mode 1 – enable transmitter data buffer mode
Return value		Transmit data buffer status (see Transmit Data Buffer Control Register on page 35 for bit definitions).

int EvrGetTxDBufStatus(volatile struct MrfErRegs *pEr);

Description		Get transmit data buffer status. When data buffer mode is enabled every other distributed bus byte is reserved for data

		transmission thus the distributed bus bandwidth is halved.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Transmit data buffer status (see Transmit Data Buffer Control Register on page 35 for bit definitions).

int EvrSendTxDBuf(volatile struct MrfErRegs *pEr, char *dbuf, int size);

Description		Get transmit data buffer status. When data buffer mode is enabled every other distributed bus byte is reserved for data transmission thus the distributed bus bandwidth is halved.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
	char *dbuf	Pointer to local data buffer
	int size	Size of data in bytes to be transmitted: 4, 8, 12, …, 2048.
Return value		Size of buffer being sent. -1 on error.

int EvrGetFormFactor(volatile struct MrfErRegs *pEr);

Description		Get form factor code from EVR.
Parameters	volatile struct MrfErRegs *pEr	Pointer to memory mapped EVR register base.
Return value		Form factor. See FPGA Firmware Version Register on page 35 for details.