

Event Generator

cPCI-EVG-220, cPCI-EVG-230

(and preliminary VME-EVG-230)

Technical Reference

Firmware Version 0001

Contents

Introduction	3
Event Stream Details	3
Event Codes	3
Distributed Bus and Data Transmission	4
Event Sources	4
Trigger Events	4
Upstream Events	5
Event Sequencer	6
Distributed Bus	7
Timestamping Inputs	8
Multiplexed Counters	8
Configurable Size Data Buffer	9
Programmable Front Panel Connections	11
AC Line Synchronisation	11
Event Clock	11
RF Clock and Event Clock	12
Fractional Synthesiser	13
Connections	13
Front Panel Connections	13
Programming Details	14
Register Map	14
Application Programming Interface (API)	28
Function Reference	28
int EvgOpen(struct MrfEgRegs **pEg, char *device_name);	28
int EvgClose(int fd);	28
int EvgEnable(volatile struct MrfEgRegs *pEg, int state);	28
int EvgGetEnable(volatile struct MrfEgRegs *pEg);	29
int EvgRxEnable(volatile struct MrfEgRegs *pEg, int state);	29
int EvgRxGetEnable(volatile struct MrfEgRegs *pEg);	29
int EvgGetViolation(volatile struct MrfEgRegs *pEg, int clear);	29
int EvgSWEventEnable(volatile struct MrfEgRegs *pEg, int state);	29

int EvgGetSWEventEnable(volatile struct MrfEgRegs *pEg);.....	30
int EvgSendSWEvent(volatile struct MrfEgRegs *pEg, int code);.....	30
int EvgEvanEnable(volatile struct MrfEgRegs *pEg, int state);.....	30
int EvgEvanGetEnable(volatile struct MrfEgRegs *pEg);.....	30
void EvgEvanReset(volatile struct MrfEgRegs *pEg);.....	30
void EvgEvanResetCount(volatile struct MrfEgRegs *pEg);.....	31
int EvgEvanGetEvent(volatile struct MrfEgRegs *pEg, struct EvanStruct *evan);	31
int EvgSetMXCPrescaler(volatile struct MrfEgRegs *pEg, int mxc, unsigned int presc);...	31
int EvgSetMxcTrigMap(volatile struct MrfEgRegs *pEg, int mxc, int map);.....	31
void EvgSyncMxc(volatile struct MrfEgRegs *pEg);.....	31
void EvgMXCDump(volatile struct MrfEgRegs *pEg);.....	32
int EvgSetDBusMap(volatile struct MrfEgRegs *pEg, int dbus, int map);	32
void EvgDBusDump(volatile struct MrfEgRegs *pEg);.....	32
int EvgSetACInput(volatile struct MrfEgRegs *pEg, int bypass, int sync, int div, int delay);	
.....	32
int EvgSetACMap(volatile struct MrfEgRegs *pEg, int map);.....	33
void EvgACDump(volatile struct MrfEgRegs *pEg);	33
int EvgSetRFInput(volatile struct MrfEgRegs *pEg, int useRF, int div);.....	33
int EvgSetFracDiv(volatile struct MrfEgRegs *pEg, int fracdiv);	33
int EvgSetSeqRamEvent(volatile struct MrfEgRegs *pEg, int ram, int pos, unsigned int	
timestamp, int code);	33
void EvgSeqRamDump(volatile struct MrfEgRegs *pEg, int ram);.....	34
int EvgSeqRamControl(volatile struct MrfEgRegs *pEg, int ram, int enable, int single, int	
recycle, int reset, int trigsel);	34
int EvgSeqRamSWTrig(volatile struct MrfEgRegs *pEg, int trig);.....	34
void EvgSeqRamStatus(volatile struct MrfEgRegs *pEg, int ram);	34
int EvgSetUnivinMap(volatile struct MrfEgRegs *pEg, int univ, int trig, int dbus);.....	35
void EvgUnivinDump(volatile struct MrfEgRegs *pEg);.....	35
int EvgSetTriggerEvent(volatile struct MrfEgRegs *pEg, int trigger, int code, int enable);	35
void EvgTriggerEventDump(volatile struct MrfEgRegs *pEg);.....	35
int EvgSetUnivOutMap(volatile struct MrfEgRegs *pEg, int output, int map);	36
int EvgSetDBufMode(volatile struct MrfEgRegs *pEg, int enable);.....	36
int EvgGetDBufStatus(volatile struct MrfEgRegs *pEg);	36
int EvgSendDBuf(volatile struct MrfEgRegs *pEg, char *dbuf, int size);	36

Introduction

The Event Generator is responsible of creating and sending out timing events to an array of Event Receivers. High configurability makes it feasible to build a whole timing system with a single Event Generator without external counters etc.

Events are sent out by the event generator as event frames (words) which consist of an eight bit event code and an eight bit distributed bus data byte. The event transfer rate is derived from an external RF clock or optionally an on-board clock generator. The optical event stream transmitted by the Event Generator is phase locked to the clock reference.

There are several sources of events: trigger events, sequence events, software events and events received from an upstream Event Generator. Events from different sources have different priority which is resolved in a priority encoder.

In addition to events the Event Generator enables the distribution of eight simultaneous signals sampled with the event clock rate, the distributed bus. Distributed bus signals may be provided externally or generated on-board by programmable multiplexed counters.

Event Stream Details

The structure of the event stream is described to help understand the functioning of the event system. The event stream should be considered as a continuous flow of event frames which consist of two bytes, the event code and distributed bus data byte.

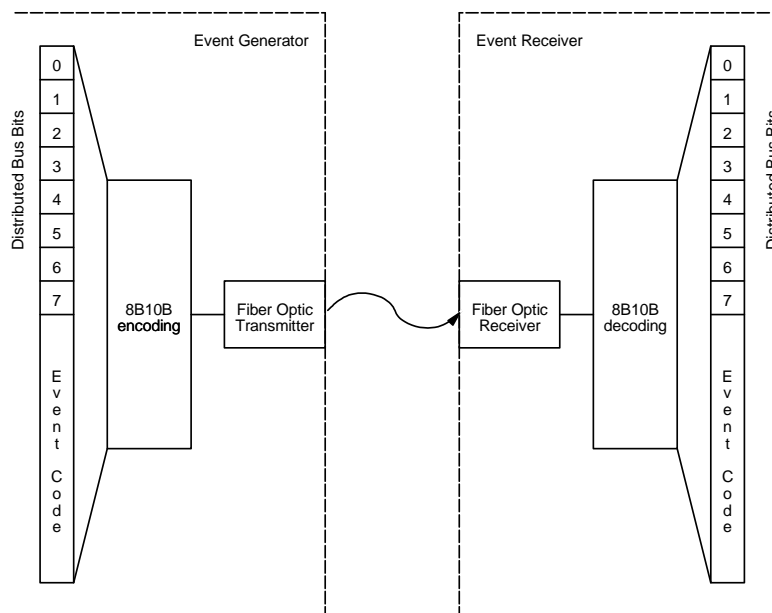


Figure 1: Event Frame

Event Codes

There are 256 event codes from which a few have special functions. The special function event codes are listed below. Only one event code may be transferred at a time. If there is no event code to be transferred, the null event code (0x00) is transmitted. Every now and then a special 8B10B

character K28.5 is transmitted instead of the null event code. The K28.5 comma character is transmitted to allow the event receivers to synchronise on the correct word boundary in the serial bit stream.

Event Code	Code Name	EVG Function	EVR Function
0x00	Null Event Code	-	-
0x01 – 0x6F	-	User Defined	User Defined
0x70	Seconds '0'	-	Shift in '0' to LSB of Seconds Shift Register
0x71	Seconds '1'	-	Shift in '1' to LSB of Seconds Shift Register
0x72 – 0x79	-	User Defined	User Defined
0x7A	Heartbeat	-	Reset Heartbeat Monitor
0x7B	Synchronise Prescalers	-	Synchronise Prescaler Outputs
0x7C	Timestamp Counter Increment	-	Increment Timestamp Counter
0x7D	Timestamp Counter Reset	-	Reset Timestamp Counter
0x7F	End of Sequence	Stop Sequence	-
0x80-FF	-	User Defined	User Defined

Distributed Bus and Data Transmission

The distributed bus allows transmission of eight simultaneous signals with the event clock rate time resolution (10 ns at 100 MHz event clock rate). The source for distributed bus signals may come from an external source or the signals may be generated with programmable multiplexed counters (MXC) inside the event generator. The distributed bus signals may be programmed to be available as hardware outputs on the event receiver.

In latest firmware versions 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 divided by two).

Event Sources

Trigger Events

There are eight trigger event sources that send out an event code on a stimulus. Each trigger event has its own programmable event code register and various enable bits. The event code transmitted is determined by contents of the corresponding event code register. The stimulus may be a detected rising edge on an external signal or a rising edge of a multiplexed counter output.

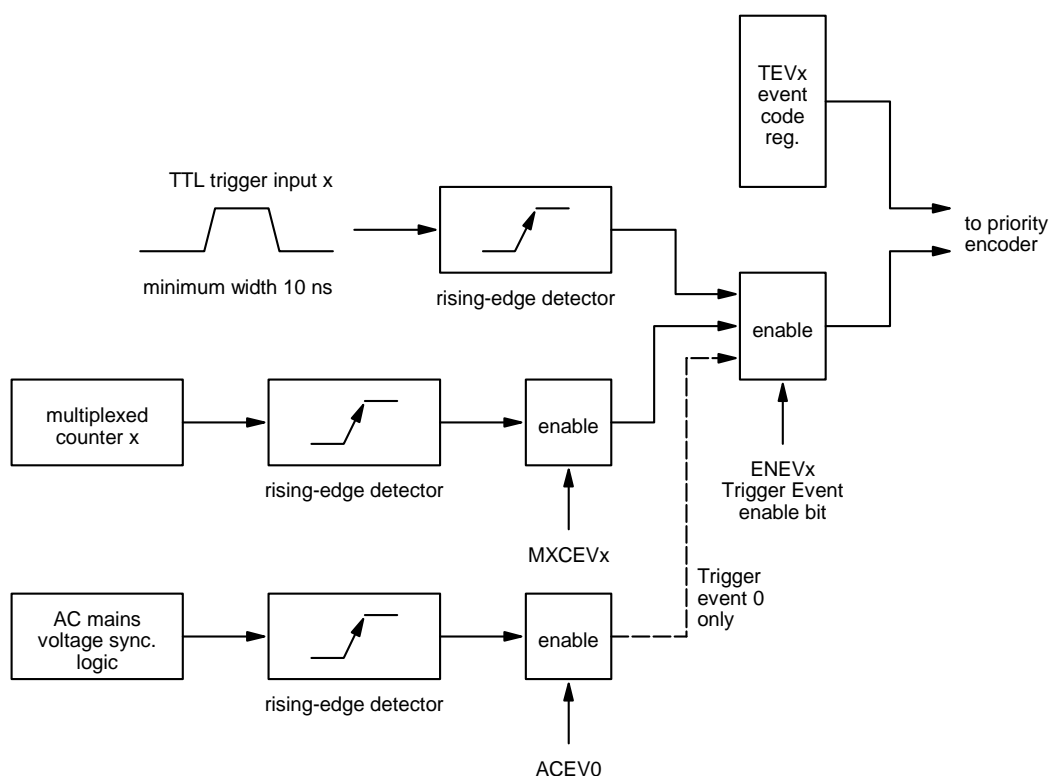


Figure 2: Trigger Events

Trigger Event 0 has also the option of being triggered by a rising edge of the AC mains voltage synchronization logic output signal.

The external input accepts TTL level signals. The input logic is edge sensitive and the signals are synchronized internally to the event clock.

Upstream Events

Event Generators may be cascaded. The event generator receiver includes a first-in-first-out (FIFO) memory to synchronize incoming events which may be synchronized to a clock unrelated to the event clock. Usually there are no events in the FIFO. An event code from an upstream EVG is transmitted as soon as there is no other event code to be transmitted.

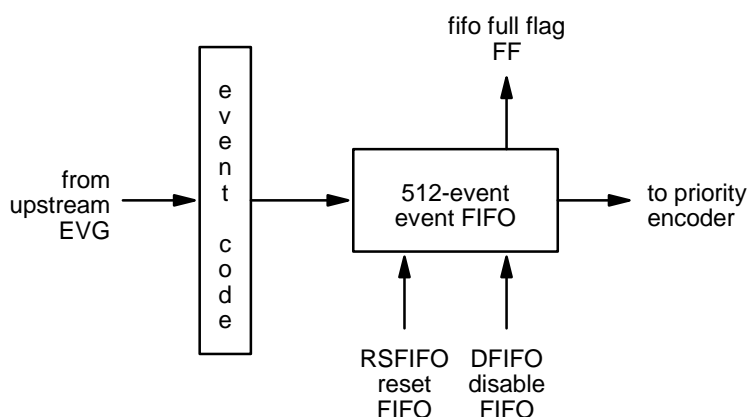


Figure 3: Upstream Event FIFO

Event Sequencer

Event sequencers provide a method of transmitting or playing back sequences of events stored in random access memory with defined timing. In the event generator there are two event sequencers. The 8-bit event codes are stored in a RAM table each attached with a 32-bit timestamp relative to the start of sequence. Both sequencers can hold up to 2048 event code – timestamp pairs.

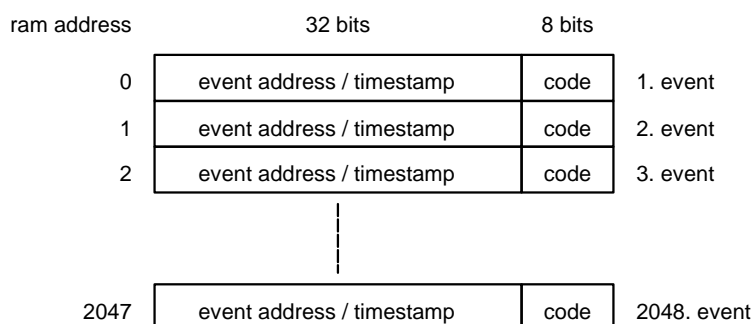


Figure 4: Sequencer RAM Structure

The contents of a sequencer RAM may be altered at any time, however, it is recommended only to modify RAM contents when the RAM is disabled. The sequencer runs at the event clock rate to up to 100 MHz.

The Sequencers may be triggered from several sources including software triggering, triggering on a multiplexed counter output or AC mains voltage synchronization logic output.

The sequencers are enabled by writing a '1' bit to SQxEN in the Sequence RAM control Register. The RAMs may be disabled any time by writing a '1' to SQxDIS bit. Disabling sequence RAMs does not reset the RAM address and timestamp registers. By writing a '1' to the bit SQxRES in the Control Register the sequencer is both disabled and the RAM address and timestamp register is reset.

When the sequencer is triggered the internal event address counters starts counting. The counter value is compared to the event address of the next event in the RAM table. When the counter value matches the timestamp in the RAM table, the attached event code is transmitted. The time

offset between two consecutive events in the RAM is allowed to be 1 to 2^{32} sequence clock cycles i.e. the internal event address counter rolls over when to 0 when 0xffffffff is reached.

There are two special event codes which are not transmitted, the null event code 0x00 and end sequence code 0x7f. The null event code may be used if the time between two consecutive events should exceed 2^{32} event clock cycles. The end sequence code resets the sequencer RAM table address and timestamp register and depending on configuration bits, disables the sequencer (single sequence, $SQxSNG=1$) or restarts the sequence either immediately (recycle sequence, $SQxREC=1$) or waits for a new trigger ($SQxREC=0$).

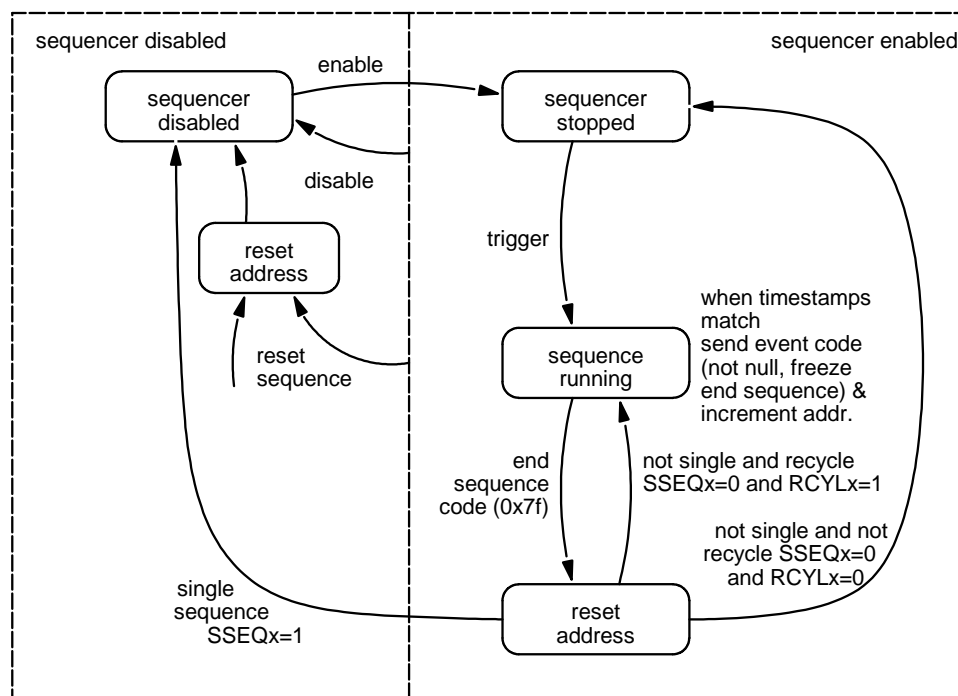


Figure 5: Sequencer Control

Distributed Bus

The bits of the distributed bus are sampled at the event rate from external signals; alternatively the distributed bus signals may be generated by multiplexed counter outputs. If there is an upstream EVG, the state of all distributed bus bits may be forwarded by the EVG.

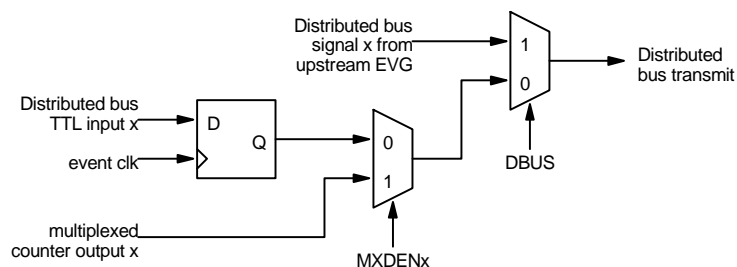


Figure 6: Distributed Bus

Timestamping Inputs

Starting from firmware version E306 a few distributed bus input signals have dual function: transition board input DBUS5-7 can be used to generate special event codes controlling the timestamping in Event Receivers.

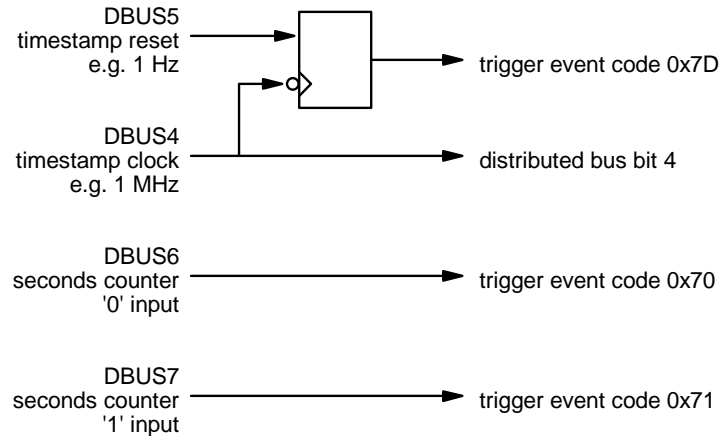


Figure 7: Timestamping Inputs

The two clocks, timestamp clock and timestamp reset clock, are assumed to be rising edge aligned. In the EVG the timestamp reset clock is sampled with the falling edge of the timestamp clock. This is to prevent a race condition between the reset and clock signals. In the EVR the reset is synchronised with the timestamp clock.

The two seconds counter events are used to shift in a 32-bit seconds value between consecutive timestamp reset events. In the EVR the value of the seconds shift register is transferred to the seconds counter at the same time the higher running part of the timestamp counter is reset.

The distributed bus event inputs can be enabled independently through the distributed bus event enable register. The events generated through these distributed bus input ports are given lowest priority.

Multiplexed Counters

Eight 32-bit multiplexed counters generate clock signals with programmable frequencies from event clock/ $2^{32}-1$ to event clock/2. Even divisors create 50% duty cycle signals. The counter outputs may be programmed to trigger events, drive distributed bus signals and trigger sequence RAMs. The output of multiplexed counter 7 is hard-wired to the mains voltage synchronization logic.

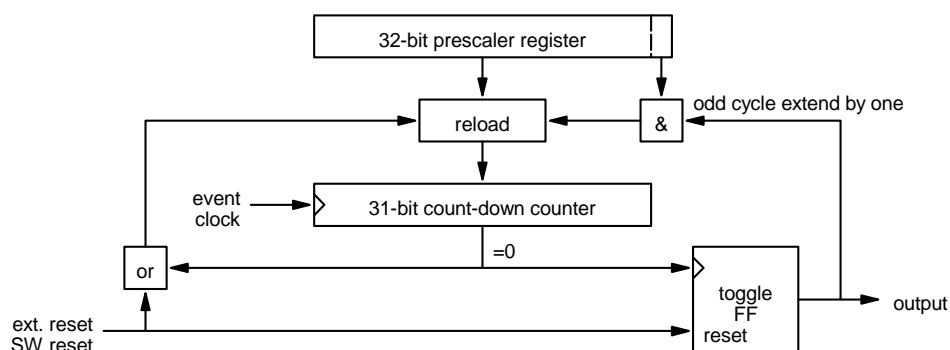


Figure 8: Multiplexed Counter

Each multiplexed counter consists of a 32-bit prescaler register and a 31-bit count-down counter which runs at the event clock rate. When count reaches zero, the output of a toggle flip-flop changes and the counter is reloaded from the prescaler register. If the least significant bit of the prescaler register is one, all odd cycles are extended by one clock cycle to support odd dividers.

Prescaler value	Duty Cycle	Frequency at 125 MHz Event Clock
0, 1 not allowed	undefined	undefined
2	50/50	62.5 MHz
3	33/66	41.7 MHz
4	50/50	31.25 MHz
5	40/60	25 MHz
...
$2^{32} - 1$	approx. 50/50	0.029 Hz

The multiplexed counters may be reset by software or hardware input. The reset state is defined by the multiplexed counter polarity register.

Configurable Size Data Buffer

Starting from firmware version E305 transmission of a configurable size data buffer over the event system link is possible. The buffer size can be programmed in four byte increments (long words) from 4 bytes to 2048 bytes.

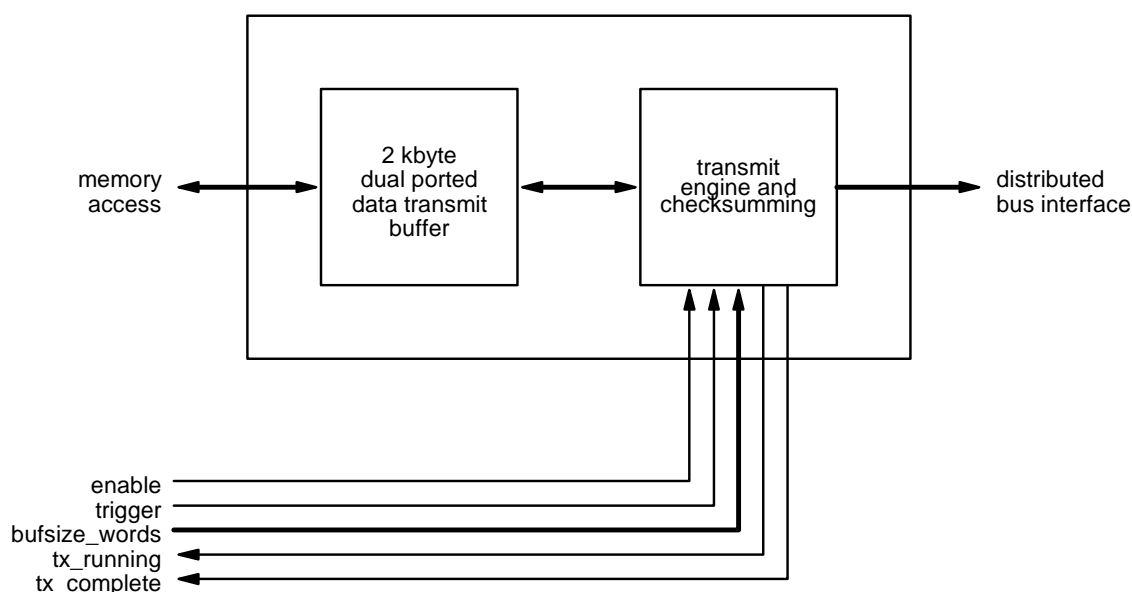


Figure 9: Configurable size transmit data buffer

When the EVG is configured for data transmission (*mode* = 1 in data buffer control register) the bandwidth of the distributed bus is shared with data transmission: half of the bandwidth remains for the distributed bus and the other half is reserved for data transmission.

The data to be transmitted is stored in a 2 kbyte dual-ported memory starting from the lowest address 0. This memory is directly accessible from VME. The transfer size is determined by *bufsize* register bits in four byte increments. The transmission is trigger by software. Two flags *tx_running* and *tx_complete* represent the status of transmission.

Transmission utilises two K-characters to mark the start and end of the data transfer payload, the protocol looks following:

Table 1: Data Transmission Protocol

8B10B-character	Description
K28.0	Start of data transfer
Dxx.x	1 st data byte (address 0)
Dxx.x	2 nd data byte (address 1)
Dxx.x	3 rd data byte (address 2)
Dxx.x	4 th data byte (address 3)
...	...
Dxx.x	n th data byte (address n-1)
K28.1	End of data
Dxx.x	Checksum (LSB)
Dxx.x	Checksum(MSB)

Programmable Front Panel Connections

The front panel outputs are programmable: multiplexed counters and distributed bus bits can be mapped to any output. The mapping is shown in table below.

Table 2: Signal mapping IDs

Mapping ID	Signal
0 to 31	(Reserved)
32	Distributed bus bit 0 (DBUS0)
...	...
39	Distributed bus bit 7 (DBUS7)
40 to 61	(Reserved)
62	Force output high (logic 1)
63	Force output low (logic 0)

AC Line Synchronisation

The Event Generator provides synchronization to the mains voltage frequency or another external clock. The mains voltage frequency can be divided by an eight bit programmable divider. The output of the divider may be delayed by 0 to 25.5 ms by a phase shifter in 0.1 ms steps to be able to adjust the triggering position relative to mains voltage phase. After this the signal synchronized to the event clock or the output of multiplexed counter 7.

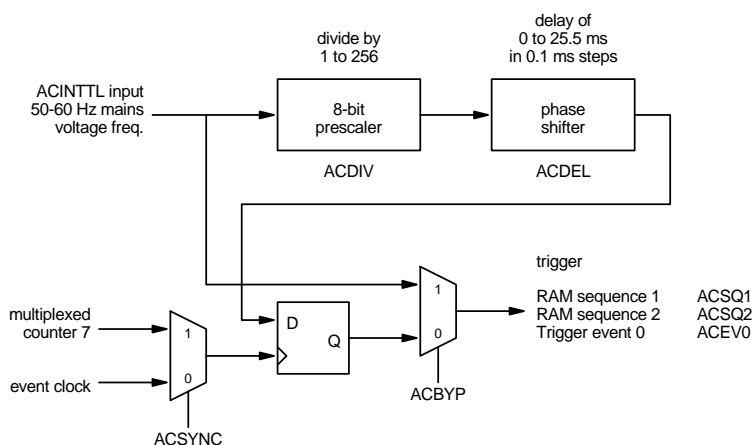


Figure 10: AC Input

The phase shifter operates with a clock of 1 MHz which introduces jitter. If the prescaler and phase shifter are not required this circuit may be bypassed. This also reduces jitter because the external trigger input is sampled directly with the event clock.

Event Clock

All operations on the event generator are synchronised to the event clock which is derived from an externally provided RF clock. For laboratory testing purposes an on-board fractional synthesiser may be used to deliver the event clock. The serial link bit rate is 20 times the event clock rate. The acceptable range for the event clock and bit rate is shown in the following table.

	Event Clock	Bit Rate
Minimum	50 MHz	1.0 Gb/s
Maximum	125 MHz	2.5 Gb/s

Note: maximum event clock for cPCI-EVG-220 is 100 MHz with 2.0 Gb/s bit rate

During operation the reference frequency should not be changed more than ± 100 ppm.

RF Clock and Event Clock

The event clock may be derived from an external RF clock signal. The front panel RF input is 50 ohm terminated and AC coupled to a LVPECL logic input, so either an ECL level clock signal or sine-wave signal with a level of maximum +10 dBm can be used.

Divider	RF Input Frequency	Event Clock	Bit Rate
$\div 1$	50 MHz – 125 MHz	50 MHz – 125 MHz	1.0 Gb/s – 2.5 Gb/s
$\div 2$	100 MHz – 250 MHz	50 MHz – 125 MHz	1.0 Gb/s – 2.5 Gb/s
$\div 3$	150 MHz – 375 MHz	50 MHz – 125 MHz	1.0 Gb/s – 2.5 Gb/s
$\div 4$	200 MHz – 500 MHz	50 MHz – 125 MHz	1.0 Gb/s – 2.5 Gb/s
$\div 5$	250 MHz – 625 MHz	50 MHz – 125 MHz	1.0 Gb/s – 2.5 Gb/s
$\div 6$	300 MHz – 750 MHz	50 MHz – 125 MHz	1.0 Gb/s – 2.5 Gb/s
$\div 7$	350 MHz – 875 MHz	50 MHz – 125 MHz	1.0 Gb/s – 2.5 Gb/s
$\div 8$	400 MHz – 1.0 GHz	50 MHz – 125 MHz	1.0 Gb/s – 2.5 Gb/s
$\div 9$	450 MHz – 1.125 MHz	50 MHz – 125 MHz	1.0 Gb/s – 2.5 Gb/s
$\div 10$	500 MHz – 1.25 GHz	50 MHz – 125 MHz	1.0 Gb/s – 2.5 Gb/s
$\div 11$	550 MHz – 1.375 GHz	50 MHz – 125 MHz	1.0 Gb/s – 2.5 Gb/s
$\div 12$	600 MHz – 1.5 GHz	50 MHz – 125 MHz	1.0 Gb/s – 2.5 Gb/s
$\div 14$	700 MHz – 1.6 GHz *)	50 MHz – 114 MHz	1.0 Gb/s – 2.286 Gb/s
$\div 15$	750 MHz – 1.6 GHz *)	50 MHz – 107 MHz	1.0 Gb/s – 2.133 Gb/s
$\div 16$	800 MHz – 1.6 GHz *)	50 MHz – 100 MHz	1.0 Gb/s – 2.0 Gb/s
$\div 17$	850 MHz – 1.6 GHz *)	50 MHz – 94 MHz	1.0 Gb/s – 1.882 Gb/s
$\div 18$	900 MHz – 1.6 GHz *)	50 MHz – 88 MHz	1.0 Gb/s – 1.777 Gb/s
$\div 19$	950 MHz – 1.6 GHz *)	50 MHz – 84 MHz	1.0 Gb/s – 1.684 Gb/s
$\div 20$	1.0 GHz – 1.6 GHz *)	50 MHz – 80 MHz	1.0 Gb/s – 1.600 Gb/s
$\div 21$	1.05 GHz – 1.6 GHz *)	50 MHz – 76 MHz	1.0 Gb/s – 1.523 Gb/s
$\div 22$	1.1 GHz – 1.6 GHz *)	50 MHz – 72 MHz	1.0 Gb/s – 1.454 Gb/s
$\div 23$	1.15 GHz – 1.6 GHz *)	50 MHz – 69 MHz	1.0 Gb/s – 1.391 Gb/s
$\div 24$	1.2 GHz – 1.6 GHz *)	50 MHz – 66 MHz	1.0 Gb/s – 1.333 Gb/s
$\div 25$	1.25 GHz – 1.6 GHz *)	50 MHz – 64 MHz	1.0 Gb/s – 1.280 Gb/s
$\div 26$	1.3 GHz – 1.6 GHz *)	50 MHz – 61 MHz	1.0 Gb/s – 1.230 Gb/s
$\div 27$	1.35 GHz – 1.6 GHz *)	50 MHz – 59 MHz	1.0 Gb/s – 1.185 Gb/s
$\div 28$	1.4 GHz – 1.6 GHz *)	50 MHz – 57 MHz	1.0 Gb/s – 1.142 Gb/s
$\div 29$	1.45 GHz – 1.6 GHz *)	50 MHz – 55 MHz	1.0 Gb/s – 1.103 Gb/s
$\div 30$	1.5 GHz – 1.6 GHz *)	50 MHz – 53 MHz	1.0 Gb/s – 1.066 Gb/s
$\div 31$	1.55 GHz – 1.6 GHz *)	50 MHz – 51 MHz	1.0 Gb/s – 1.032 Gb/s
$\div 32$	1.6 GHz *)	50 MHz	1.0 Gb/s

*) Range limited by AD9515 maximum input frequency of 1.6 GHz

Note: maximum event clock for cPCI-EVG-220 is 100 MHz with 2.0 Gb/s bit rate

Fractional Synthesiser

For laboratory testing purposes the event clock may be generated on-board the event generator 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/4 = 124.95 MHz	0x00FE816D	124.95 MHz	0
499.654 MHz/4 = 124.9135 MHz	0x0C928166	124.907 MHz	-52 ppm
476 MHz/4 = 119 MHz	0x018741AD	119 MHz	0
106.25 MHz (fibre channel)	0x049E81AD	106.25 MHz	0
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
499.654 MHz/4 = 124.9135 MHz	0x0C928166	124.907 MHz	-52 ppm
50 MHz	0x009743AD	50.0 MHz	0

Connections

Front Panel Connections

The front panel of the Event Generator and its optional side-by-side module is shown in Figure 11 and Figure 12.

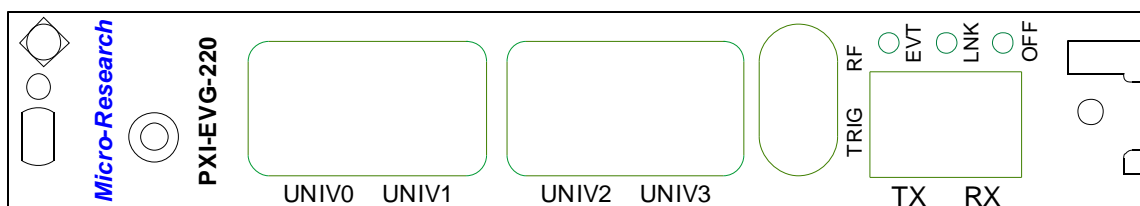


Figure 11: Event Generator Front Panel

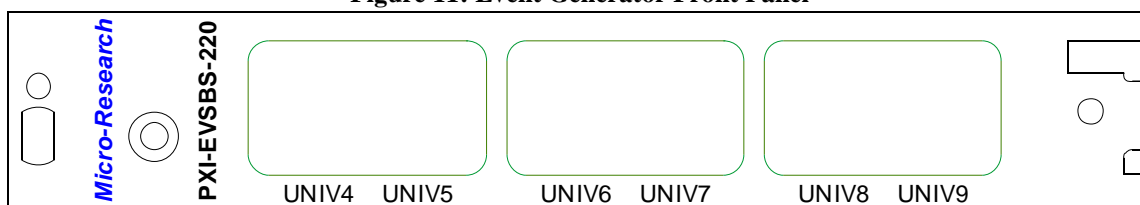


Figure 12: Optional Side-by-side Module Front Panel

The front panel of the Event Generator 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)
RF	LEMO-EPY	RF	RF/event clock input
TRIG	LEMO-EPY	TTL	AC Trigger input
UNIV0/1	Universal slot		Universal Input 0/1
UNIV2/3	Universal slot		Universal Input 2/3
UNIV4/5	Universal slot		Universal Input 4/5
UNIV6/5	Universal slot		Universal Input 6/7
UNIV8/9	Universal slot		Universal Input 8/9

Programming Details

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	ACControl	UINT32	AC divider control
0x014	ACMap	UINT32	AC trigger event mapping
0x018	SWEvent	UINT32	Software event register
0x020	DataBufControl	UINT32	Data Buffer Control Register
0x024	DBusMap	UINT32	Distributed Bus Mapping Register
0x02C	FWVersion	UINT32	Firmware Version Register
0x04C	UsecDivider	UINT32	Divider to get from Event Clock to 1 MHz
0x050	ClockControl	UINT32	Event Clock Control Register
0x060	EvanControl	UINT32	Event Analyser Control Register
0x064	EvanCode	UINT32	Event Analyser Distributed Bus and Event Code Register
0x068	EvanTimeHigh	UINT32	Event Analyser Time Counter (bits 63 – 32)
0x06C	EvanTimeLow	UINT32	Event Analyser Time Counter (bits 31 – 0)
0x070	SeqRamCtrl0	UINT32	Sequence RAM 0 Control Register
0x074	SeqRamCtrl1	UINT32	Sequence RAM 1 Control Register
0x080	FracDiv	UINT32	Micrel SY87739L Fractional Divider Configuration Word

0x100	EvTrig0	UINT32	Event Trigger 0 Register
0x104	EvTrig1	UINT32	Event Trigger 1 Register
0x108	EvTrig2	UINT32	Event Trigger 2 Register
0x10C	EvTrig3	UINT32	Event Trigger 3 Register
0x110	EvTrig4	UINT32	Event Trigger 4 Register
0x114	EvTrig5	UINT32	Event Trigger 5 Register
0x118	EvTrig6	UINT32	Event Trigger 6 Register
0x11C	EvTrig7	UINT32	Event Trigger 7 Register
0x180	MXCCtrl0	UINT32	Multiplexed Counter 0 Control Register
0x184	MXCPresc0	UINT32	Multiplexed Counter 0 Prescaler Register
0x188	MXCCtrl1	UINT32	Multiplexed Counter 1 Control Register
0x18C	MXCPresc1	UINT32	Multiplexed Counter 1 Prescaler Register
0x190	MXCCtrl2	UINT32	Multiplexed Counter 2 Control Register
0x194	MXCPresc2	UINT32	Multiplexed Counter 2 Prescaler Register
0x198	MXCCtrl3	UINT32	Multiplexed Counter 3 Control Register
0x19C	MXCPresc3	UINT32	Multiplexed Counter 3 Prescaler Register
0x1A0	MXCCtrl4	UINT32	Multiplexed Counter 4 Control Register
0x1A4	MXCPresc4	UINT32	Multiplexed Counter 4 Prescaler Register
0x1A8	MXCCtrl5	UINT32	Multiplexed Counter 5 Control Register
0x1AC	MXCPresc5	UINT32	Multiplexed Counter 5 Prescaler Register
0x1B0	MXCCtrl6	UINT32	Multiplexed Counter 6 Control Register
0x1B4	MXCPresc6	UINT32	Multiplexed Counter 6 Prescaler Register
0x1B8	MXCCtrl7	UINT32	Multiplexed Counter 7 Control Register
0x1BC	MXCPresc7	UINT32	Multiplexed Counter 7 Prescaler Register
0x400	FPOutMap0	UINT16	Front Panel Output 0 Mapping Register
0x402	FPOutMap1	UINT16	Front Panel Output 1 Mapping Register
0x404	FPOutMap2	UINT16	Front Panel Output 2 Mapping Register
0x406	FPOutMap3	UINT16	Front Panel Output 3 Mapping Register
0x440	UnivOutMap0	UINT16	Universal Output 0 Mapping Register
0x442	UnivOutMap1	UINT16	Universal Output 1 Mapping Register
0x444	UnivOutMap2	UINT16	Universal Output 2 Mapping Register
0x446	UnivOutMap3	UINT16	Universal Output 3 Mapping Register
0x448	UnivOutMap4	UINT16	Universal Output 4 Mapping Register
0x44A	UnivOutMap5	UINT16	Universal Output 5 Mapping Register
0x44C	UnivOutMap6	UINT16	Universal Output 6 Mapping Register
0x44E	UnivOutMap7	UINT16	Universal Output 7 Mapping Register
0x450	UnivOutMap8	UINT16	Universal Output 8 Mapping Register
0x452	UnivOutMap9	UINT16	Universal Output 9 Mapping Register
0x500	FPInMap0	UINT32	Front Panel Input 0 Mapping Register
0x504	FPInMap1	UINT32	Front Panel Input 1 Mapping Register
0x540	UnivInMap0	UINT32	Front Panel Universal Input 0 Map Register
0x544	UnivInMap1	UINT32	Front Panel Universal Input 1 Map Register
0x548	UnivInMap2	UINT32	Front Panel Universal Input 2 Map Register
0x54C	UnivInMap3	UINT32	Front Panel Universal Input 3 Map Register

0x550	UnivInMap4	UINT32	Front Panel Universal Input 4 Map Register
0x554	UnivInMap5	UINT32	Front Panel Universal Input 5 Map Register
0x558	UnivInMap6	UINT32	Front Panel Universal Input 6 Map Register
0x55C	UnivInMap7	UINT32	Front Panel Universal Input 7 Map Register
0x560	UnivInMap8	UINT32	Front Panel Universal Input 8 Map Register
0x564	UnivInMap9	UINT32	Front Panel Universal Input 9 Map Register
0x600	TBInMap0	UINT32	Transition Board Input 0 Mapping Register
0x604	TBInMap1	UINT32	Transition Board Input 1 Mapping Register
0x608	TBInMap2	UINT32	Transition Board Input 2 Mapping Register
0x60C	TBInMap3	UINT32	Transition Board Input 3 Mapping Register
0x610	TBInMap4	UINT32	Transition Board Input 4 Mapping Register
0x614	TBInMap5	UINT32	Transition Board Input 5 Mapping Register
0x618	TBInMap6	UINT32	Transition Board Input 6 Mapping Register
0x61C	TBInMap7	UINT32	Transition Board Input 7 Mapping Register
0x620	TBInMap8	UINT32	Transition Board Input 8 Mapping Register
0x624	TBInMap9	UINT32	Transition Board Input 9 Mapping Register
0x628	TBInMap10	UINT32	Transition Board Input 10 Mapping Register
0x62C	TBInMap11	UINT32	Transition Board Input 11 Mapping Register
0x630	TBInMap12	UINT32	Transition Board Input 12 Mapping Register
0x634	TBInMap13	UINT32	Transition Board Input 13 Mapping Register
0x638	TBInMap14	UINT32	Transition Board Input 14 Mapping Register
0x63C	TBInMap15	UINT32	Transition Board Input 15 Mapping Register
0x800 – 0xFFF	DataBuf		Data Buffer Transmit Memory
0x8000 – 0xBFFF	SeqRam0		Sequence RAM 0
0xC000 – 0xFFFF	SeqRam1		Sequence RAM 1

Status Register

address	bit 31	bit 30	bit 29	Bit 28	bit 27	bit 26	bit 25	bit 24
0x000	RDB7	RDB6	RDB5	RDB4	RDB3	RDB2	RDB1	RDB0

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x001	TDB7	TDB6	TDB5	TDB4	TDB3	TDB2	TDB1	TDB0

Bit	Function
RDB7	Status of received distributed bus bit 7 (from upstream EVG)
RDB6	Status of received distributed bus bit 6 (from upstream EVG)
RDB5	Status of received distributed bus bit 5 (from upstream EVG)
RDB4	Status of received distributed bus bit 4 (from upstream EVG)
RDB3	Status of received distributed bus bit 3 (from upstream EVG)
RDB2	Status of received distributed bus bit 2 (from upstream EVG)
RDB1	Status of received distributed bus bit 1 (from upstream EVG)
RDB0	Status of received distributed bus bit 0 (from upstream EVG)

TDB7	Status of transmitted distributed bus bit 7
TDB6	Status of transmitted distributed bus bit 6
TDB5	Status of transmitted distributed bus bit 5
TDB4	Status of transmitted distributed bus bit 4
TDB3	Status of transmitted distributed bus bit 3
TDB2	Status of transmitted distributed bus bit 2
TDB1	Status of transmitted distributed bus bit 1
TDB0	Status of transmitted distributed bus bit 0

Control Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x004	EVGEN	RXDIS	RXPWD					MXCRES

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x005								SRALT

Bit	Function
EVGEN	Event Generator Master enable
RXDIS	Disable event reception
RXPWD	Receiver Power down
MXCRES	Write 1 to reset multiplexed counters
SRALT	(reserved)

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			IFDBUF				IFFF	IFVIO

Bit	Function
IFDBUF	Data buffer flag
IFFF	RX 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			IEDBUF				IEFF	IEVIO

Bit	Function
IRQEN	Master interrupt enable
IEDBUF	Data buffer interrupt enable
IEFF	Event FIFO full interrupt enable

IEVIO Receiver violation interrupt enable

AC Trigger Control Register

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x011							ACBYP	ACSYNC

address	bit 15	Bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x012	AC Trigger Divider							

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x013	AC Trigger Phase Shift							

Bit	Function
ACBYP	AC divider and phase shifter bypass (0 = divider/phase shifter enabled, 1 = divider/phase shifter bypassed)
ACSYNC	Synchronization select (0 = event clock, 1 = multiplexed counter 7 output)

AC Trigger Mapping Register

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x017	ACM7	ACM6	ACM5	ACM4	ACM3	ACM2	ACM1	ACM0

Bit	Function
ACM7	If set AC circuit triggers Event Trigger 7
ACM6	If set AC circuit triggers Event Trigger 6
ACM5	If set AC circuit triggers Event Trigger 5
ACM4	If set AC circuit triggers Event Trigger 4
ACM3	If set AC circuit triggers Event Trigger 3
ACM2	If set AC circuit triggers Event Trigger 2
ACM1	If set AC circuit triggers Event Trigger 1
ACM0	If set AC circuit triggers Event Trigger 0

Software Event Register

address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x01A							SWPEND	SWENA

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x01B	Event Code to be sent out							

Bit	Function
SWPEND	Event code waiting to be sent out (read-only). A new event code may be written to the event code register when this bit reads '0'.
SWENA	Enable software event

Data Buffer Control Register

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x021				TXCPT	TXRUN	TRIG	ENA	MODE

address	bit 15	Bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x022						DTSZ(10:8)		

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x023	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
ENA	Write ‘1’ to start transmission of data in buffer 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

Distributed Bus Mapping Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x024	DBMAP7(3:0)				DBMAP6(3:0)			

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x025	DBMAP5(3:0)				DBMAP4(3:0)			

address	bit 15	Bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x026	DBMAP3(3:0)				DBMAP2(3:0)			

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x027	DBMAP1(3:0)				DBMAP0(3:0)			

Bits	Function
DBMAP7(3:0)	Distributed Bus Bit 7 Mapping: 0 – Off, output logic ‘0’ 1 – take bus bit from external input 2 – Multiplexed counter output mapped to distributed bus bit 3 – Distributed bus bit forwarded from upstream EVG
DBMAP6(3:0)	Distributed Bus Bit 7 Mapping (see above for mappings)
DBMAP5(3:0)	Distributed Bus Bit 7 Mapping (see above for mappings)
DBMAP4(3:0)	Distributed Bus Bit 7 Mapping (see above for mappings)
DBMAP3(3:0)	Distributed Bus Bit 7 Mapping (see above for mappings)
DBMAP2(3:0)	Distributed Bus Bit 7 Mapping (see above for mappings)

DBMAP1(3:0) Distributed Bus Bit 7 Mapping (see above for mappings)

DBMAP0(3:0) Distributed Bus Bit 7 Mapping (see above for mappings)

FPGA Firmware Version Register

address	bit 31	bit 27	bit 26	bit 24
0x02C	EVG = 0x2			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

Microsecond Divider Register

address	bit 15	bit 0
0x04e	Rounded integer value of 1 μ s * 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 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x050								EXTRF

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x051			RFDIV5	RFDIV4	RFDIV3	RFDIV2	RFDIV1	RFDIV0

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

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

Bit	Function
RFSEL5-0	External RF divider select: 000000 – RF/1 000001 – RF/2 000010 – RF/3 000011 – RF/4 000100 – RF/5 000101 – RF/6

000110 – RF/7
 000111 – RF/8
 001000 – RF/9
 001001 – RF/10
 001010 – RF/11
 001011 – RF/12
 001100 – OFF
 001101 – RF/14
 001110 – RF/15
 001111 – RF/16
 010000 – RF/17
 010001 – RF/18
 010010 – RF/19
 010011 – RF/20
 010100 – RF/21
 010101 – RF/22
 010110 – RF/23
 010111 – RF/24
 011000 – RF/25
 011001 – RF/26
 011010 – RF/27
 011011 – RF/28
 011100 – RF/29
 011101 – RF/30
 011110 – RF/31
 011111 – RF/32

EXTRF RF reference select:
 0 – Use internal reference (fractional synthesizer)
 1 – Use external RF reference
 CGLOCK Micrel SY87739L locked (read-only)

Event Analyser Control Register

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x063				EVANE	EVARS	EVAOF	EVAEN	EVACR

Bits	Function
EVANE	Event Analyser FIFO not empty flag: 0 – FIFO empty 1 – FIFO not empty, events in FIFO
EVARS	Event Analyser Reset 0 – not in reset 1 – reset
EVAOF	Event Analyser FIFO overflow flag: 0 – no overflow 1 – FIFO overflow

EVAEN	Event Analyser enable 0 – Event Analyser disabled 1 – Event Analyser enabled
EVACR	Event Analyser 64 bit counter reset 0 – Counter running 1 – Counter reset to zero.

Event Analyser Data Register

address	bit 15	bit 8	bit 7	bit 0
0x066	(reserved)			Event Code

Event Analyser Counter Registers

address	bit 31	bit 0
0x068	Event Analyser Counter Register (bits 63 – 32)	

address	bit 31	bit 0
0x06C	Event Analyser Counter Register (bits 31 – 0)	

Sequence RAM Control Registers

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x070							SQ0RUN	SQ0ENA

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x071			SQ0SWT	SQ0SNG	SQ0REC	SQ0RES	SQ0DIS	SQ0EN

address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x072								

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x073	SQ0TSEL							

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x074							SQ1RUN	SQ1ENA

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x075			SQ1SWT	SQ1SNG	SQ1REC	SQ1RES	SQ1DIS	SQ1EN

Address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x076								

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x077	SQ1TSEL							

Bit	Function
SQxRUN	Sequence RAM running flag (read-only)
SQxENA	Sequence RAM enabled flag (read_only)

SQxSWT	Sequence RAM software trigger, write '1' to trigger
SQxSNG	Sequence RAM single mode
SQxREC	Sequence RAM recycle mode
SQxRES	Sequence RAM reset, write '1' to reset
SQxDIS	Sequence RAM disable, write '1' to disable
SQxEN	Sequence RAM enable, write '1' to enable/arm
SQxTSEL	Sequence RAM trigger select: 0 – trigger from MXC0 1 – trigger from MXC1 2 – trigger from MXC2 3 – trigger from MXC3 4 – trigger from MXC4 5 – trigger from MXC5 6 – trigger from MXC6 7 – trigger from MXC7 16 – trigger from AC synchronization logic 17 – trigger from sequence RAM 0 software trigger 18 – trigger from sequence RAM 1 software trigger

SY87739L Fractional Divider Configuration Word

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

Configuration Word	Frequency with 24 MHz reference oscillator
0x0C928166	124.907 MHz
0x0C9282A6	62.454 MHz
0x009743AD	50 MHz
0xC25B43AD	49.978 MHz
0x0176C36D	49.965 MHz

Event Trigger Registers

Address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x102								EVEN0

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x103	EVCD0(7:0)							

Address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x106								EVEN1

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x107	EVCD1(7:0)							

Address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x10A								EVEN2

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x10B	EVCD2(7:0)							

Address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x10E								EVEN3

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x10F	EVCD3(7:0)							

Address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x102								EVEN4

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x103	EVCD4(7:0)							

Address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x106								EVEN5

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x107	EVCD5(7:0)							

Address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x10A								EVEN6

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x10B	EVCD6(7:0)							

Address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x10E								EVEN7

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x10F	EVCD7(7:0)							

Bit	Function
EVENx	Enable Event Trigger x
EVCDx	Event Trigger Code for Event trigger x

Multiplexed Counter Registers

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x180	MXC0	MXP0						

address	Bit 7	bit 6	bit 5	Bit 4	Bit 3	bit 2	bit 1	bit 0
0x183	MX0EV7	MX0EV6	MX0EV5	MX0EV4	MX0EV3	MX0EV2	MX0EV1	MX0EV0

address	bit 31	bit 0
0x184	Multiplexed Counter 0 prescaler	

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x188	MXC1	MXP1						

address	Bit 7	bit 6	bit 5	Bit 4	Bit 3	bit 2	bit 1	bit 0
0x18B	MX1EV7	MX1EV6	MX1EV5	MX1EV4	MX1EV3	MX1EV2	MX1EV1	MX1EV0

address	bit 31	bit 0
0x18C	Multiplexed Counter 1 prescaler	

Address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x190	MXC2	MXP2						

Address	Bit 7	bit 6	bit 5	Bit 4	Bit 3	bit 2	bit 1	bit 0
0x193	MX2EV7	MX2EV6	MX2EV5	MX2EV4	MX2EV3	MX2EV2	MX2EV1	MX2EV0

address	bit 31	bit 0
0x194	Multiplexed Counter 2 prescaler	

Address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x198	MXC3	MXP3						

Address	Bit 7	bit 6	bit 5	Bit 4	Bit 3	bit 2	bit 1	bit 0
0x19B	MX3EV7	MX3EV6	MX3EV5	MX3EV4	MX3EV3	MX3EV2	MX3EV1	MX3EV0

address	bit 31	bit 0
0x19C	Multiplexed Counter 3 prescaler	

Address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x1A0	MXC4	MXP4						

Address	Bit 7	bit 6	bit 5	Bit 4	Bit 3	bit 2	bit 1	bit 0
0x1A3	MX4EV7	MX4EV6	MX4EV5	MX4EV4	MX4EV3	MX4EV2	MX4EV1	MX4EV0

address	bit 31	Bit 0
0x1A4	Multiplexed Counter 4 prescaler	

Address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	Bit 24
0x1A8	MXC5	MXP5						

Address	Bit 7	bit 6	bit 5	Bit 4	Bit 3	bit 2	bit 1	bit 0
0x1AB	MX5EV7	MX5EV6	MX5EV5	MX5EV4	MX5EV3	MX5EV2	MX5EV1	MX5EV0

address	bit 31	bit 0
0x1AC	Multiplexed Counter 5 prescaler	

Address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x1B0	MXC6	MXP6						

address	Bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
0x441		Universal I/O OUT0 Mapping ID (see Table 2 for mapping IDs)						
0x443				Universal I/O OUT1 Mapping ID				
0x445				Universal I/O OUT2 Mapping ID				
0x447				Universal I/O OUT3 Mapping ID				
0x449				Universal I/O OUT4 Mapping ID				
0x44B				Universal I/O OUT5 Mapping ID				
0x44D				Universal I/O OUT6 Mapping ID				
0x44F				Universal I/O OUT7 Mapping ID				
0x451				Universal I/O OUT8 Mapping ID				
0x453				Universal I/O OUT9 Mapping ID				

Notes:

cPCI-EVG has a maximum of four Universal I/O outputs and six additional outputs are provided by the optional side-by-side module.
VME-EVG-230 has a maximum four Universal I/O outputs.

Front Panel Input Mapping Registers

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x501	FP0DB7	FP0DB6	FP0DB5	FP0DB4	FP0DB3	FP0DB2	FP0DB1	FP0DB0

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x503	FP0EV7	FP0EV6	FP0EV5	FP0EV4	FP0EV3	FP0EV2	FP0EV1	FP0EV0

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x505	FP1DB7	FP1DB6	FP1DB5	FP1DB4	FP1DB3	FP1DB2	FP1DB1	FP1DB0

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x507	FP1EV7	FP1EV6	FP1EV5	FP1EV4	FP1EV3	FP1EV2	FP1EV1	FP1EV0

Bit Function

FPxDB7	Map Front panel Input x to Distributed Bus bit 7
FPxDB6	Map Front panel Input x to Distributed Bus bit 6
FPxDB5	Map Front panel Input x to Distributed Bus bit 5
FPxDB4	Map Front panel Input x to Distributed Bus bit 4
FPxDB3	Map Front panel Input x to Distributed Bus bit 3
FPxDB2	Map Front panel Input x to Distributed Bus bit 2
FPxDB1	Map Front panel Input x to Distributed Bus bit 1
FPxDB0	Map Front panel Input x to Distributed Bus bit 0
FPxEV7	Map Front panel Input x to Event Trigger 7
FPxEV6	Map Front panel Input x to Event Trigger 6
FPxEV5	Map Front panel Input x to Event Trigger 5
FPxEV4	Map Front panel Input x to Event Trigger 4
FPxEV3	Map Front panel Input x to Event Trigger 3
FPxEV2	Map Front panel Input x to Event Trigger 2
FPxEV1	Map Front panel Input x to Event Trigger 1
FPxEV0	Map Front panel Input x to Event Trigger 0

Universal Input Mapping Registers

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x541	UI0DB7	UI0DB6	UI0DB5	UI0DB4	UI0DB3	UI0DB2	UI0DB1	UI0DB0

Address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x543	UI0EV7	UI0EV6	UI0EV5	UI0EV4	UI0EV3	UI0EV2	UI0EV1	UI0EV0

Bit Function

UIxDB7	Map Universal Input x to Distributed Bus bit 7
UIxDB6	Map Universal Input x to Distributed Bus bit 6
UIxDB5	Map Universal Input x to Distributed Bus bit 5
UIxDB4	Map Universal Input x to Distributed Bus bit 4
UIxDB3	Map Universal Input x to Distributed Bus bit 3

UIxDB2	Map Universal Input x to Distributed Bus bit 2
UIxDB1	Map Universal Input x to Distributed Bus bit 1
UIxDB0	Map Universal Input x to Distributed Bus bit 0
UIxEV7	Map Universal Input x to Event Trigger 7
UIxEV6	Map Universal Input x to Event Trigger 6
UIxEV5	Map Universal Input x to Event Trigger 5
UIxEV4	Map Universal Input x to Event Trigger 4
UIxEV3	Map Universal Input x to Event Trigger 3
UIxEV2	Map Universal Input x to Event Trigger 2
UIxEV1	Map Universal Input x to Event Trigger 1
UIxEV0	Map Universal Input x to Event Trigger 0

Note: all enabled input signals are OR'ed together. So if e.g. distributed bus bit 0 has two sources from universal input 0 and 1, if either of the inputs is active high also the distributed bus is active high.

Application Programming Interface (API)

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

Function Reference

int EvgOpen(struct MrfEgRegs **pEg, char *device_name);

Description	Opens the EVG device for access. Simultaneous accesses are allowed.				
Parameters	<table><tr><td>struct MrfEgRegs **pEg</td><td>EvgOpen returns pointer to EVG registers by memory mapping the I/O registers into user space.</td></tr><tr><td>char *device_name</td><td>Holds the device name of the EVG, e.g. /dev/ega3. The device names are set up by the module_load script of the device driver.</td></tr></table>	struct MrfEgRegs **pEg	EvgOpen returns pointer to EVG registers by memory mapping the I/O registers into user space.	char *device_name	Holds the device name of the EVG, e.g. /dev/ega3. The device names are set up by the module_load script of the device driver.
struct MrfEgRegs **pEg	EvgOpen returns pointer to EVG registers by memory mapping the I/O registers into user space.				
char *device_name	Holds the device name of the EVG, 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 EvgClose(int fd);

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

int EvgEnable(volatile struct MrfEgRegs *pEg, int state);

Description	Enables the EVG and allows sending event codes.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base.

int state

0: disable

1: enable

Return value

Returns zero when EVG disabled

Returns non-zero when EVG enabled

int EvgGetEnable(volatile struct MrfEgRegs *pEg);**Description**

Retrieves state of the EVG.

Parameters

volatile struct MrfEgRegs *pEg

Pointer to memory mapped EVG register base.

Return value

Returns zero when EVG disabled

Returns non-zero when EVG enabled

int EvgRxEnable(volatile struct MrfEgRegs *pEg, int state);**Description**

Enables/disables the EVG receiver.

Parameters

volatile struct MrfEgRegs *pEg

Pointer to memory mapped EVG register base.

int state

0: disable

1: enable

Return value

Returns zero when RX disabled

Returns non-zero when RX enabled

int EvgRxGetEnable(volatile struct MrfEgRegs *pEg);**Description**

Retrieves state of the EVG receiver.

Parameters

volatile struct MrfEgRegs *pEg

Pointer to memory mapped EVG register base.

Return value

Returns zero when RX disabled

Returns non-zero when RX enabled

int EvgGetViolation(volatile struct MrfEgRegs *pEg, int clear);**Description**

Get/clear EVG RX link violation status.

Parameters

volatile struct MrfEgRegs *pEg

Pointer to memory mapped EVG 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.

int EvgSWEventEnable(volatile struct MrfEgRegs *pEg, int state);**Description**

Enable sending of software event codes.

Parameters

volatile struct MrfEgRegs *pEg

Pointer to memory mapped EVG register base.

int state

0: disable

1: enable

Return value

Returns zero when EVG SW events disabled

Returns non-zero when EVG SW events enabled

int EvgGetSWEventEnable(volatile struct MrfEgRegs *pEg);

Description	Retrieve state of software event codes.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base.
Return value	Returns zero when EVG SW events disabled Returns non-zero when EVG SW events enabled

int EvgSendSWEvent(volatile struct MrfEgRegs *pEg, int code);

Description	Send software event code.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base. int code Event code to be sent out
Return value	Returns code sent out.

int EvgEvanEnable(volatile struct MrfEgRegs *pEg, int state);

Description	Enable/disable EVG event analyzer.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base. int state 0: disable 1: enable
Return value	Returns zero when EVG event analyzer disabled Returns non-zero when EVG SW event analyzer enabled

int EvgEvanGetEnable(volatile struct MrfEgRegs *pEg);

Description	Get EVG event analyzer state.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base.
Return value	Returns zero when EVG event analyzer disabled Returns non-zero when EVG SW event analyzer enabled

void EvgEvanReset(volatile struct MrfEgRegs *pEg);

Description	Reset EVG event analyzer state.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base.
Return value	none

void EvgEvanResetCount(volatile struct MrfEgRegs *pEg);

Description	Reset EVG event analyzer time counter value.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base.
Return value	None

int EvgEvanGetEvent(volatile struct MrfEgRegs *pEg, struct EvanStruct *evan);

Description	Retrieve one event from event analyzer.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base. struct EvanStruct *evan Pointer to event analyzer structure to store one event. (see egapi.h for structure details).
Return value	Returns zero on success. Returns -1 if no events available in event analyzer.

int EvgSetMXCPrescaler(volatile struct MrfEgRegs *pEg, int mxc, unsigned int presc);

Description	Set multiplexed counter prescaler.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base. int mxc Multiplexed counter number 0-7. unsigned int presc 32-bit prescaler value.
Return value	Returns zero on success. Returns -1 on error.

int EvgSetMxcTrigMap(volatile struct MrfEgRegs *pEg, int mxc, int map);

Description	Set multiplexed counter to event trigger mapping.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base. int mxc Multiplexed counter number 0-7. int map Number of event trigger to map to.
Return value	Returns zero on success. Returns -1 on error.

void EvgSyncMxc(volatile struct MrfEgRegs *pEg);

Description	Synchronize multiplexed counters.
--------------------	-----------------------------------

Parameters	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.
Return value		None

void EvgMXCDump(volatile struct MrfEgRegs *pEg);

Description		Dump multiplexed counter registers.
Parameters	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.
Return value		None

int EvgSetDBusMap(volatile struct MrfEgRegs *pEg, int dbus, int map);

Description		Set distributed bus bit mappings.
Parameters	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.
	int dbus	Distributed bus bit number 0-7.
	int map	Distributed bus bit source: C_EVG_DBUS_SEL_OFF: bit tied to zero C_EVG_DBUS_SEL_EXT: external input C_EVG_DBUS_SEL_MXC: multiplexed counter C_EVG_DBUS_SEL_FORWARD: from upstream EVG
Return value		Returns zero on success. Returns -1 on error.

void EvgDBusDump(volatile struct MrfEgRegs *pEg);

Description		Dump distributed bus registers.
Parameters	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.
Return value		None

int EvgSetACInput(volatile struct MrfEgRegs *pEg, int bypass, int sync, int div, int delay);

Description		Set AC input parameters.
Parameters	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.
	int bypass	0: use AC sync logic 1: bypass phase shifter and divider
	int sync	0: don't synchronize to MXC7 1: synchronize to MXC7
	int div	Divider 1 – 255
	int delay	Phase shift in approx. 0.1 ms steps
Return value		Returns zero on success.

Returns -1 on error.

int EvgSetACMap(volatile struct MrfEgRegs *pEg, int map);

Description	Set AC input event trigger mapping.				
Parameters	<table><tr><td>volatile struct MrfEgRegs *pEg</td><td>Pointer to memory mapped EVG register base.</td></tr><tr><td>int map</td><td>Number of event trigger to map to.</td></tr></table>	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.	int map	Number of event trigger to map to.
volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.				
int map	Number of event trigger to map to.				
Return value	Returns zero on success. Returns -1 on error.				

void EvgACDump(volatile struct MrfEgRegs *pEg);

Description	Dump AC input registers.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base.
Return value	None

int EvgSetRFInput(volatile struct MrfEgRegs *pEg, int useRF, int div);

Description	Set up event clock RF input.						
Parameters	<table><tr><td>volatile struct MrfEgRegs *pEg</td><td>Pointer to memory mapped EVG register base.</td></tr><tr><td>int useRF</td><td>0: use internal reference (fractional synthesizer) 1: use external RF input</td></tr><tr><td>int div</td><td>C_EVG_RFDIV_1, C_EVG_RFDIV_2, etc. see egapi.h for details.</td></tr></table>	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.	int useRF	0: use internal reference (fractional synthesizer) 1: use external RF input	int div	C_EVG_RFDIV_1, C_EVG_RFDIV_2, etc. see egapi.h for details.
volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.						
int useRF	0: use internal reference (fractional synthesizer) 1: use external RF input						
int div	C_EVG_RFDIV_1, C_EVG_RFDIV_2, etc. see egapi.h for details.						
Return value	Returns zero on success. Returns -1 on error.						

int EvgSetFracDiv(volatile struct MrfEgRegs *pEg, int fracdiv);

Description	Set fractional divider control word which provides reference frequency for receiver.				
Parameters	<table><tr><td>volatile struct MrfEgRegs *pEg</td><td>Pointer to memory mapped EVG register base.</td></tr><tr><td>int fracdiv</td><td>Fractional divider control word</td></tr></table>	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.	int fracdiv	Fractional divider control word
volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.				
int fracdiv	Fractional divider control word				
Return value	Returns control word written				

int EvgSetSeqRamEvent(volatile struct MrfEgRegs *pEg, int ram, int pos, unsigned int timestamp, int code);

Description	Write one event into Sequence RAM.				
Parameters	<table><tr><td>volatile struct MrfEgRegs *pEg</td><td>Pointer to memory mapped EVG register base.</td></tr><tr><td>int ram</td><td>Number of Sequence RAM</td></tr></table>	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.	int ram	Number of Sequence RAM
volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.				
int ram	Number of Sequence RAM				

	0: RAM0
	1: RAM1
int pos	Event position in memory: 0 – 2047
unsigned int timestamp	Timestamp of event (32-bit)
int code	Event code (8-bit)
Return value	Returns zero on success. Returns -1 on error.

void EvgSeqRamDump(volatile struct MrfEgRegs *pEg, int ram);

Description	Dump Sequence RAM registers.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base.
Return value	None

int EvgSeqRamControl(volatile struct MrfEgRegs *pEg, int ram, int enable, int single, int recycle, int reset, int trigsel);

Description	Setup Sequence RAM
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base.
	int ram Number of Sequence RAM
	0: RAM0
	1: RAM1
	int enable 0: disable RAM
	1: enable RAM
	int single 0: multi-sequence
	1: single sequence
	int recycle 0: trigger mode
	1: recycle mode (loop)
	int reset 1: reset RAM
	int trigsel See egapi.h
Return value	Returns zero on success. Returns -1 on error.

int EvgSeqRamSWTrig(volatile struct MrfEgRegs *pEg, int trig);

Description	Software trigger Sequence RAM.
Parameters	volatile struct MrfEgRegs *pEg Pointer to memory mapped EVG register base.
	int trig 0: software trigger 0
	1: software trigger 1
Return value	Returns 0 on success. Returns -1 on error.

void EvgSeqRamStatus(volatile struct MrfEgRegs *pEg, int ram);

Description	Dump Sequence RAM status.
--------------------	---------------------------

Parameters	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.
Return value		None

int EvgSetUnivinMap(volatile struct MrfEgRegs *pEg, int univ, int trig, int dbus);

Description		Set up universal input mappings.
Parameters	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.
	int univ	Number of universal input (0-3 for EVG, 4-9 for side-by-side module)
	int trig	Number of event trigger to map to.
	int dbus	Number of external distributed bus input to map to.
Return value		Returns 0 on success. Returns -1 on error.

void EvgUnivinDump(volatile struct MrfEgRegs *pEg);

Description		Dump Universal input mappings.
Parameters	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.
Return value		None

int EvgSetTriggerEvent(volatile struct MrfEgRegs *pEg, int trigger, int code, int enable);

Description		Set up trigger events.
Parameters	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.
	int trigger	Number of trigger event
	int code	Event code
	int enable	0: disable 1: enable
Return value		Returns 0 on success. Returns -1 on error.

void EvgTriggerEventDump(volatile struct MrfEgRegs *pEg);

Description		Dump Event trigger settings.
Parameters	volatile struct MrfEgRegs *pEg	Pointer to memory mapped EVG register base.
Return value		None

int EvgSetUnivOutMap(volatile struct MrfEgRegs *pEg, int output, int map);

Description

Set up universal output mappings.

Parameters volatile struct MrfEgRegs *pEg

Pointer to memory mapped EVG register base.

int output

Universal Output number

int map

Signal mapping (see egapi.h for details)

Return value

Returns 0 on success, -1 on error

int EvgSetDBufMode(volatile struct MrfEgRegs *pEg, 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 MrfEgRegs *pEg

Pointer to memory mapped EVG register base.

int enable

0 – disable transmitter data buffer mode

1 – enable transmitter data buffer mode

Return value

Transmit data buffer status (see **Error! Reference source not found.** on page **Error! Bookmark not defined.** for bit definitions).

int EvgGetDBufStatus(volatile struct MrfEgRegs *pEg);

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 MrfEgRegs *pEg

Pointer to memory mapped EVG register base.

Return value

Transmit data buffer status (see **Error! Reference source not found.** on page **Error! Bookmark not defined.** for bit definitions).

int EvgSendDBuf(volatile struct MrfEgRegs *pEg, 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 MrfEgRegs *pEg

Pointer to memory mapped EVG register base.

char *dbuf
int size

Return value

Pointer to local data buffer
Size of data in bytes to be transmitted:
4, 8, 12, ..., 2048.
Size of buffer being sent.
-1 on error.