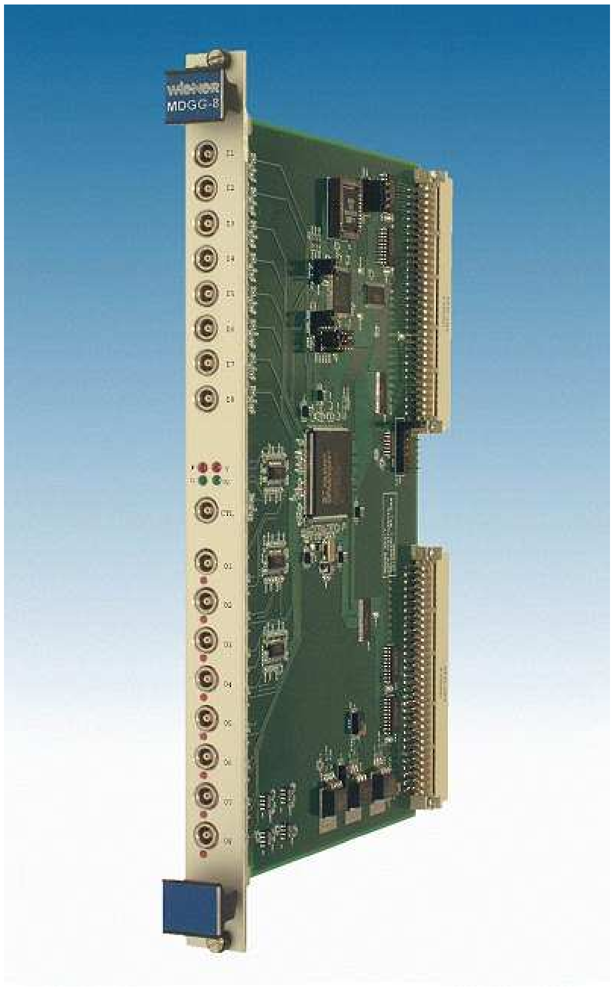
MDGG-8



User Manual

General Remarks

The only purpose of this manual is a description of the product. It must not be interpreted a declaration of conformity for this product including the product and software.
W-Ie-Ne-R revises this product and manual without notice. Differences of the description in manual and product are possible.
W-Ie-Ne-R excludes completely any liability for loss of profits, loss of business, loss of use or data, interrupt of business, or for indirect, special incidental, or consequential damages of any kind, even if **W-Ie-Ne-R** has been advises of the possibility of such damages arising from any defect or error in this manual or product.
Any use of the product which may influence health of human beings requires the express written permission of **W-Ie-Ne-R**.
Products mentioned in this manual are mentioned for identification purposes only.
Product names appearing in this manual may or may not be registered trademarks or copyrights of their respective companies.
No part of this product, including the product and the software may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means with the express written permission of **W-Ie-Ne-R**.

MDGG-8 is designed by JTEC Instruments.

Table of contents:

1 GENERAL DESCRIPTION

The MDGG-8 is a single width VME module performing multifunction gate and delay generator and logic functions. Most firmware revisions will allow users to create delay and gate generators, count triggers, perform logic AND's, and digital FAN-IN / FAN-OUT. This manual will strive to describe the functionality of the of the latest firmware revision. Details about how particular firmware revisions differ can be found in the appendix for that specific revision.

1.1 Hardware features

Although the functionality of the MDGG-8 is determined by the firmware that is loaded, the firmware capabilities must fall within the parameters of the hardware. The MDGG-8 is based on an XCS3S400 XLINX FPGA running with a 125MHz clock. The module consists of:

INPUTS:	8 channel inputs + 1 common (LEMO 00)
OUTPUTS:	8 channel outputs (LEMO 00)
LEDS:	12 Diagnostic LED's, driven by signal stretchers
FIRMWARE:	Programmed via VME
INTERFACE:	VME A24 D32, base address via jumpers.
VME IRQ:	Capable of asserting VME IRQ, level selected via jumper
POWER:	5V, 1A

1.2 Release firmware features (preliminary)

The MDGG-8 can hold two separate firmware revisions in memory. The user selects which firmware will load at startup via a jumper of the PCB. The latest firmware revsions can be found at http://www.wiener-d.com. Because of the open architecture of the module, users are invited to develop custom firmware.

As of this writing, the MDGG-8 ships with firmware revision 7930 0100. This firmware provides:

- **8-Flexible Gate and Delay Generators (FGG)**

  Up to eight FGG's are individually configurable logic signals with flexible gates and delays. They are programmable as non-triggerable delay and gate generators (DGG), delay and retriggerable gate (RDGG), set-resest gates (SRG) or pulse generators (PG). Alternatively, the FGG's can be set as1/n (PSG) or 1-1/n (CPSG) prescaler gates. The gates and delays are set with 8ns granularity up to 34s each.

- **8-Scaler Devices**

  The eight scaler devices can be configured as either a gated scaler or a latched scaler. Gated scalers (GSC) provide 32-bits of storage for each channel. The latched scalers (LSC) provide each have a 1kx32-bit FIFO for storing the scaler data.

- **Coincidence Register**

  There is one 8-bit coincidence register to do something. This register will record input coincidences.

- **4-Combined Gates**

  There are up to four combined gates (CG) available for two-fold Ors or eight-fold ANDs of the module inputs. This allows the user to build complex triggers.

- **Output Mulitplexing**

  The eight outputs can be multiplexed to have any one of 20 signals, choosing from the 8 FGGs output signals, trailing edge of the 8 FGG signals, or the 4 combined gates.

- **Diagnostic LEDs**
  - LED 1: FPGA status:
    - ON- FPGA not configured
    - Flashing- flash memory being configured
    - OFF- MDGG-8 configured

- LED 2: DTACK, Indicates VME access
- LED 3:Indicates signal on COMMON input
- LED 4: Indicates signal on one or more of I1-I8
- LED 5-12: Indicates signal on the corresponding output

- **VETO/Reset input**
  The common input can be configured as a VETO signal to block any of the eight regular inputs or as a reset, to clear or latch the internal scalers.

- **VME Resets and Triggering**
  The internal scaler devices can be reset via VME. Additionally, the FGGs can be triggered or SRG can be reset through the VME interface.

2 VME INTERFACE

The MDGG-8 is access via the VME bus using A24D32 read/writes or BLT reads.

2.1 VME Base Address

The MDGG-8 base address is set via jumpers on the PCB. A jumper that is inserted counts as a 1 in the base address bit pattern.

A23	A22	A21	A20	A19	A18	A17	A16
SN0	SN1	SN2	SN3	SN4	SN5	0	0

2.2 Firmware programming

It is possible to load the FPGA firmware via the on-board JTEG interface.

2.3 VME Register Map

Offset	Register	Access Type
0x000	Firmware ID	Read only
0x004	Global Register	Read/write
0x008	Auxiliary Register	Read/write
0x040 + channel * 8	Delay FGG (channel)	Read/write
0x044 + channel * 8	Gate FGG (channel)	Read/write
0x080	Action Register	Write only
0x084	FGG Configuration	Read/write
0x088	Scaler Configuration	Read/write
0x08C	FGG Input Selector A	Read/write
0x090	FGG Input Selector B	Read/write
0x094	NIM Output Selector A	Read/write
0x098	NIM Output Selector B	Read/write
0x09C	FGG Stop Selector A	Read/write
0x0A0	FGG Stop Selector B	Read/write
0x0A4	Scaler Input Selector A	Read/write
0x0A8	Scaler Input Selector B	Read/write
0x0AC	Combination Gate Mask A	Read/write
0x0B0	Combination Gate Mask B	Read/write
0x100 + channel * 4	Scaler Data (channel)	Read only
0x120 + channel * 4	Scaler Latch Multiplicity (channel)	Read only
0x140	Coincidence Register	Read only

2.3.1 Firmware ID Register [% +0x000, RO]

The Firmware ID register holds the firmware revision identifier.

2.3.2 Global Register [% +0x004 RW]

The Global register stores information for the global signals that affect all different devices that the MDGG-8 is capable of providing.

Bits 24 – 28	Bits 16 - 20	Bits 8 – 12	Bits 0 – 4
SelMReset	**SelSclrLatch**	**SelSclrGa**	**SelFGGVeto**

SelFGG VETO: This value specifies which signal or input will be used to VETO all FGGs that are configured to be susceptible to VETOs.

SelSclrGa: This value specifies which signal or input will be used as the gate for internal scalers that are configured as gated scalers.

SelSclrLatch: This value specifies which signal or input will be used as the latching signal for internal scalers that are configured as latched scalers.

SelMReset: This value specifies which signal or input will be used as the master reset used to reset SRG, scaler, and the coincidence registers.

The valid signals for each of the settings in the Global Register are represented by a 5-bit code. These codes are shown in the table below.

Values	Signal	Comments
0	None selected	No signal will perform that function
1-8	I1-I8	1-8 corresponds to I1-I8 respectively
9-16	FGG1-FGG8	9-16 corresponds to FGG1-FGG8 respectively
17-20	CG1-CG4	17-20 corresponds to CG1-CG4 respectively
21	Common Input	The common input will serve the function

2.3.3 Auxiliary Register [% +0x008, RW]

The Auxiliary Register determines which FGGs are cleared upon a master reset signal, which signal is used to clear the coincidence register, and whether the coincidence register is cleared upon a master reset signal.

Bits 13	Bits 8 - 12	Bits 0 – 7
CRegMResetMask	**SelCRegGa**	**FGGMResetMask**

FGGMResetMask: This 8-bit mask determines which FGG channels will be reset by a master reset signal

SelCRegGa: This value specifies which signal or input will be used to clear the coincidence register. Valid values are:

Code	Signal
0-7	I1-I8
8-15	FGG1-FGG8
16-19	CG1-CG-4
20	Common Input

CRegMResetMask: If this bit is set, the coincidence register will be reset upon a master reset signal being received.

2.3.4 Delay FGG [% +0x040 + 8 * channel, RW]

The Delay FGG registers store the 32-bit delay between receiving a trigger and the start of the gate. There are eight Delay FGG registers, each corresponding to one of eight FGG Gate registers. The delay can be set with 8ns granularity with a maximum of 34s.

For a given delay time, t_{delay}, the register value, $r_{value,}$ can be calculated by:

$$r_{value} = \frac{t_{delay}}{8*10^{-9}}$$

For given register value, r_{value}, the delay time, $t_{delay,}$ can be calculated by:

$$t_{gate} = r_{value} *8*10^{-9}$$

2.3.5 Gate FGG [% +0x044 + 8 * channel, RW]

The Gate FGG registers store the 32-bit gate length. There are eight Gate FGG registers, each corresponding to one of eight FGG Delay registers. The Gate can be set with 8ns granularity with a maximum of 34s.

For a given gate length, $t_{gate,}$ the register value, $r_{value,}$ can be calculated by:

$$r_{value} = \frac{t_{gate}}{8*10^{-9}}$$

For given register value, r_{value}, the gate length, $t_{gate,}$ can be calculated by:

$$t_{gate} = r_{value} *8*10^{-9}$$

2.3.6 Action Register [% +0x080, WO]

The Action Register allows the user to trigger FGGs, reset SRGs, reset scalers, and/or reset the coincidence register. The action register automatically clears all bits 16ns, after they are set.

Bit 24	Bits 16 - 23	Bits 8 - 15	Bits 0 - 7
ResCREG	**TrigFGG1-8**	**ResSclr1-8**	**ResFGG1-8**

ResFGG1-8: Bits 0-7 correspond to FGG1-FGG-8 respectively. Writing “1” to the bit corresponding to a given FGG channel will reset that SRG.

ResSclr1-8: Bits 8-15 correspond to SCL1-SCL-8 respectively. Writing “1” to the bit corresponding to a given scaler channel, will reset that scaler.

ResFGG1-8: Bits 16-23 correspond to FGG1-FGG-8 respectively. Writing “1” to the bit corresponding to a given FGG channel will trigger that FGG.

ResCReg: Writing “1” to bit 24 of the Action Register will reset the Coincidence register.

2.3.7 FGG Configuration [% +0x080, RW]

The eight FGG devices can be configured as a non-retriggerable DGG, a set-reset gate, a pulser, a retriggable DGG, a 1/n prescaler, or a complimentary 1-1/n prescaler. Additionally, each FGG can be configured to be blocked via the VETO signal. The settings for each FGG is stored in the FGG Configuration register.

The settings for each FGG is set in a 4-bit configuration word.

3	Bits 0-2
VETO	**Behavior**

Valid values for the Behavior bits are:

Value	Behavior	Description
0	FGG off	FGG will not respond to triggers
1	DGG	FGG will act as a standard DGG that retriggers every time a trigger is received.
2	SRG	FGG will start the gate. The gate will stay set until the reset signal is received.
3	PG	FGG will act as a pulser, with the period equal to gate + delay.
4	RDGG	FGG will act as a retriggable DGG. The gate will be extended by the gate length if a new trigger arrives while the gate is still active.
5	PSG	FGG will act as a 1/n prescaler. I need to research how exactly this will work.
6	CPSG	FGG will act as a 1-1/n prescaler. This will make more sense after figuring out the PSG.

If bit-3 of the configuration word is set, the FGG will not respond to triggers while a VETO signal is present.

The configuration words for each FGG are combined in the FGG configuration register.

28-31	24-27	20-23	16-19	12-15	8-11	4-7	Bits 0-3
FGG8	**FGG7**	**FGG6**	**FGG5**	**FGG4**	**FGG3**	**FGG2**	**FGG1**

2.3.8 Scaler Configuration [% +0x088, RW]

The Scaler Configuration allows the user to set each scaler as a gated, latched, or gated-latched scaler.

The settings for each scaler are set via a 4-bit configuration word.

3	Bits 0-2
MReset	**Behavior**

Valid values for the Behavior bits are:

Value	Behavior	Description
0	Scaler Off	Scaler will not count

1	Gated Scaler	Scaler will count triggers anytime the gated is active and will suspend counting when the gate is inactive. The count is stored in a 32-bit register.
2	Latched Scaler	Scaler will count until a latch signal is received. The count will be placed in 1kx32-bit FIFO and a new count will begin.
3	Gated – Latched Scaler	Scaler will act as a latched scaler but will suspend count counting when the gate is inactive.

If bit-3 of the configuration word is set, the scaler count will be reset via a master reset trigger.

The configuration words for each scaler are combined in the scaler configuration register.

28-31	**24-27**	**20-23**	**16-19**	**12-15**	**8-11**	**4-7**	**Bits 0-3**
SCLR8	**SCLR7**	**SCLR6**	**SCLR5**	**SCLR4**	**SCLR3**	**SCLR2**	**SCLR1**

2.3.9 FGG Input Selector Registers [% +0x08C, 0x090, RW]

There are two FGG Input Selector Registers which are used to determine which signal triggers the start of a FGG. Each FGG Input Selector Register sets the input trigger for 4 FGGs.

The input for each FGG is specified by a 5-bit configuration word which specifies which input or signal will be used to trigger the gate. The table below shows the valid values for the configuration word and which correspond to each triggerable signal.

Code	**Trigger Signal**
0-7	I1-I8
8-15	FGG1-FGG8
16-19	CG1-CG-4
20	Common Input

The FGG input selection configuration words are combined into the two FGG input selector registers.

FGG Input Selector A [% +0x08C, RW]

Bits24-28	**Bits 16 – 20**	**Bits 8 – 12**	**Bits 0 - 4**
TrigSelFGG4	**TrigSelFGG3**	**TrigSelFGG2**	**TrigSelFGG1**

FGG Input Selector B [% +0x090, RW]

Bits24-28	**Bits 16 – 20**	**Bits 8 – 12**	**Bits 0 - 4**
TrigSelFGG8	**TrigSelFGG7**	**TrigSelFGG6**	**TrigSelFGG5**

2.3.10 NIM Output Selector Registers [% +0x08C, 0x090, RW]

There are two NIM output Selector Registers which are used to determine what signal is present on each of the NIM outputs, O1-O8. Each NIM Ouput Selector Register sets 4 outputs.

The signal for each NIM output is specified by a 5-bit configuration word which specifies which signal will be present on the NIM connector. The table below shows the valid values for the configuration word and which code corresponds to each available signal.

Code	Output Signal	Comments
0-7	FGG1-FGG8	The output will be a FGG signal
8-15	TrEdg1-TrEdg8	The output will be a signal on the trailing edge of FGG1-FGG8
16-19	CG1-CG-4	The output will be one of the combined gates, CG1-CG2

The NIM output selection configuration words are combined into the two NIM output selector registers.

NIM Output Selector A [% +0x094, RW]

Bits24-28	Bits 16 – 20	Bits 8 – 12	Bits 0 - 4
NIMOutSel 4	NIMOutSel 3	NIMOutSel 2	NIMOutSel1

NIM Output Selector B [% +0x098, RW]

Bits24-28	Bits 16 – 20	Bits 8 – 12	Bits 0 - 4
NIMOutSel 8	NIMOutSel 7	NIMOutSel 6	NIMOutSel 5

2.3.11 FGG Stop Selector Registers [% +0x094, 0x098, RW]

There are two Stop Selector Registers which are used to determine what signal resets an FGG that is set as set-reset gate (SRG). A FGG set as a SRG will start the gate upon a trigger signal (as set in the FGG Input selector registers). That gate will remain active until a stop signal is received. The stop signal can come from a master reset (subject to the FGGMResetMask in the Auxiliary Register), a VME reset via the Action Register, or from a signal specified in the FGG Stop Selector Register.

The reset for each FGG is specified by a 5-bit configuration word which specifies which signal will reset a given SRG. The table below shows the valid values for the configuration word and which code corresponds to each available signal.

Code	Reset Signal	Comments
0-7	I1-I2	The reset will be performed upon a signal in the corresponding input.
8-15	TrEdg1-TrEdg8	The reset be performed on the trailing edge of FGG1-FGG8.
16-19	CG1-CG-4	The reset will be performed upon signal from one of the combined gates, CG1-CG2.

20	Common Input	The rest will performed upon signal from the common input.

The FGG Stop selection configuration words are combined into the two FGG Stop selector registers.

FGG Stop Selector A [% +0x09C, RW]

Bits24-28	Bits 16 – 20	Bits 8 – 12	Bits 0 - 4
NIMOutSel 4	NIMOutSel 3	NIMOutSel 2	NIMOutSel1

FGG Stop Selector B [% +0x0A0, RW]

Bits24-28	Bits 16 – 20	Bits 8 – 12	Bits 0 - 4
NIMOutSel 8	NIMOutSel 7	NIMOutSel 6	NIMOutSel 5

2.3.12 Scaler Input Selector Registers [% +0x0A4, 0x0A8, RW]

There are two Scaler Input Selector Registers which are used to determine which trigger signal is counted by a particular scaler. Each FGG Input Selector Register sets the input trigger for 4 FGGs.

The input for each scaler is specified by a 5-bit configuration word which specifies which input or signal will be counted. The table below shows the valid values for the configuration word and which corresponds to each countable signal.

Code	Trigger Signal
0-7	I1-I8
8-15	FGG1-FGG8
16-19	CG1-CG-4
20	Common Input

The FGG input selection configuration words are combined into the two FGG input selector registers.

Scaler Input Selector A [% +0x0A4, RW]

Bits24-28	Bits 16 - 20	Bits 8 – 12	Bits 0 - 4
SclrInSel 4	SclrInSel 3	SclrInSel 2	SclrInSel 1

Scaler Input Selector B [% +0x0A8, RW]

Bits24-28	Bits 16 – 20	Bits 8 – 12	Bits 0 - 4
NIMOutSel 8	NIMOutSel 7	NIMOutSel 6	NIMOutSel 5

2.3.13 Combination Gate Mask Registers [% +0x0AC, 0x0B0, RW]

There are two Combination Gate Mask Registers which are used to setup the 4-combination gates (CG) in the MDGG-8. The combination gates allow the user to trigger a gate based on a combination of logic ANDs and Ors of the input signals (I1-I8). Each

CG has two 8-bit mask words that ware used to determine whether to trigger the gate using the following logic equation:

CGn = {[AMASK(n,1) AND NIM}OR {[AMASK(n,2) AND NIM]}

where NIM is an 8-bit word representing the status of the 8 NIM inputs.
The mask words for two CGs are combined in each Combination Gate Mask Register.

Combination Gate Mask A [% +0x0AC, RW]

Bits24-31	Bits 16 - 23	Bits 8 – 15	Bits 0 – 7
AMASK(2,2)	**AMASK(2,1)**	**AMASK(1,2)**	**AMASK(1,1)**

Combination Gate Mask A [% +0x0B0, RW]

Bits24-31	Bits 16 - 23	Bits 8 – 15	Bits 0 – 7
AMASK(4,2)	**AMASK(4,1)**	**AMASK(3,2)**	**AMASK(3,1)**

2.3.14 Scaler Data Registers [% +0x100 + 4 * channel, RO]

For each scaler, a 1kx32-bit FIFO exists to store the scaler data. The data is read via a VME call to the Scaler data registers. The depth of the FIFO (number values in the FIFO) for each scaler channel can be read from the corresponding Scaler Latch Multiplicity Register. For scalers operated only in gated mode, the depth of the FIFO is always 1. Gated scalers must be reset before being able to accept a subsequent gate.

The Scaler Data Register can be read out via VME block transfers (BLT).

2.3.15 Scaler Multiplicity Registers [% +0x120 + 4 * channel, RO]

The scaler multiplicity register stores the depth (number of 32-bit words) that are stored in the Scaler Data FIFO for each scaler.

2.3.16 Coincidence Register [% +0x120 + 4 * channel, RO]

The Coincidence register shows the coincidence pattern of I1-I8 during a gate. This pattern can be read out via VME and must be cleared before it will accept the next gate.

3 SOFTWARE SUPPORT

WIENER supplies software support for the MDGG-8 when it is used in combination with the VM-USB, VME crate controller.

3.1 Microsoft Windows Support

WIENER support of the MDGG-8 under Microsoft Windows comes in two forms. The first is an open source C++ DLL which will provide an easy way to develop custom applications for the MDGG-8 and VM-USB. Additionally, a prepackaged executable is available for setting up and reading the MDGG-8. This executable is an extension of the XXUSBwin program shipped with the VM-USB.

3.2 Linux Support

WIENER supports Linux users by providing an open source C++ library to allow for easy setup and readout of the MDGG-8 when used in combination with the WIENER VM-USB.

3.3 MDGG-8 Library

The WIENER supplied open source library has the same structure for both Windows and Linux users. While the library is designed to work with the WIENER VM-USB, users should be able to modify to work with their VME crate controller as well since the source code is available.

The CMDGG-8 library provides the CMDGG-8 class for performing all of the necessary tasks for setting up and reading an MDGG-8 module.

3.3.1 CMDGG

The CMDGG is the constructor of the MDGG-8 class. It creates a CMDGG object that all the member function will act upon.

```
CMDGG{
        usb_dev_handle *hdev,
        unsigned long base
};
```

Parameters

**hdev*

[in] Pointer to the VM-USB device handle. (Returned by xx_usb_device_open from the xx_usb library.)

Base

[in] Base address of the MDGG-8 you wish to control.

Return Value
Returns a MDGG object.

Remarks

3.3.2 getFirmware

The getFirmware function will return the contents of the MDGG-8 firmware revision register.

unsigned long getFirmware{
};

Parameters:

Return Value
On success, the content of the MDGG-8 firmware revision register is returned. Otherwise, the return value is 0.

Remarks

3.3.3 getGlobalRegister

The getGlobalRegister function will return the content of the MDGG-8 global register.

unsigned long getGlobalRegister{
};

Parameters:

Return Value
On success, the content of the MDGG-8 global register is returned. Otherwise, the return value is 0.

Remarks

3.3.4 setGlobalRegister

The setGlobalRegister function will write value into the global register of the MDGG-8.

int setGlobalRegister{
unsigned long value

};

Parameters:
value
[in] value that will be written to the MDGG-8 global register.

Return Value
On success, the return value will be > 0. Otherwise the value will be <1.

Remarks

3.3.5 setGate

The setGate function writes a gate value to the specified FGG gate register.

int setGate{
int channel,
unsigned long gate
};

Parameters:
channel
[in] the FGG channel whose gate should be set (valid values 1-8)
gate
[in] the length to which the gate should be set in terms of 8ns steps.

Return Value
On Success, the return value will be >0. Otherwise the return value will be <1.

Remarks

3.3.6 getGate

The getGate function returns the gate of the specified FGG gate register.

unsigned long getGate{
int channel
};

Parameters:
channel
[in] the FGG channel for which you wish to return the gate value.

Return Value

On success, the value of the specified gate register is returned. Otherwise, the return value is 0.

Remarks
The gate value is returned as it is stored in the gate register. To obtain the gate length, multiply the non-zero return value by 8ns.

3.3.7 setDelay

The setDelay function writes a delay value to the specified FGG delay register.

```
int setDelay{
        int channel,
        unsigned long delay
};
```

Parameters:
channel
[in] the FGG channel whose gate should be set (valid values 1-8)
delay
[in] the length to which the delay should be set in terms of 8ns steps.

Return Value
On Success, the return value will be >0. Otherwise the return value will be <1.

Remarks

3.3.8 getDelay

The getDelay function returns the delay of the specified FGG delay register.

```
unsigned long getDelay{
        int channel
};
```

Parameters:
channel
[in] the FGG channel for which you wish to return the delay value.

Return Value
On success, the value of the specified delay register is returned. Otherwise, the return value is 0.

Remarks

The delay value is returned as it is stored in the delay register. To obtain the delay length, multiply the non-zero return value by 8ns.

3.3.9 setActionRegister

The setActionRegister function writes a specified value to the action register.

int setActionRegister {
        unsigned long value
};

Parameters:
value
        [in] the value to write to the action register.

Return Value
On Success, the return value will be >0. Otherwise the return value will be <1.

Remarks
The action register acts as a momentary switch, when a value written to the register, some action is performed and the register is cleared in the next FPGA cycle.

3.3.10 setFGGConfiguration

The setFGGConfiguration function writes a specified value to the FGG configuration register.

int setFGGConfiguration {
        unsigned long value
};

Parameters:
value
        [in] the value to write to the FGG configuration. Register.

Return Value
On Success, the return value will be >0. Otherwise the return value will be <1.

Remarks

3.3.11 getFGGConfiguration

The getFGGConfiguration function returns the value in the FGG Configuration register.

unsigned long getFGGConfiguration{
};

Parameters:

Return Value
On success, the value of the FGG Configuration register is returned. Otherwise, the return value is 0.

Remarks

3.3.12 setScalerConfiguration

The setScalerConfiguration function writes a specified value to the Scaler configuration register.

int setScalerConfiguration {
    unsigned long value
};

Parameters:
value
    [in] the value to write to the Scaler configuration register.

Return Value
On Success, the return value will be >0. Otherwise the return value will be <1.

Remarks

3.3.13 getScalerConfiguration

The getFGGConfiguration function returns the value in the FGG Configuration register.

unsigned long getScalerConfiguration{
};

Parameters:

Return Value
On success, the value of the ScalerConfiguration register is returned. Otherwise, the return value is 0.

Remarks

3.3.14 setFGGInputSelector

The setFGGInputSelector function writes a given value to the specified FGGInputSelector register.

```
int setFGGInputSelector {
        int reg,
        unsigned long value
};
```

Parameters:

Reg

[in] which FGGInputSelector register to write (1 or 2).

value

[in] the value to write to the FGGInputSelector register.

Return Value

On Success, the return value will be >0. Otherwise the return value will be <1.

Remarks

3.3.15 getFGGInputSelector

The getFGGInputSelector function returns the value of the specified FGGInputSelector register.

```
unsigned long getFGGInputSelector{
        int reg,
};
```

Parameters:

Reg

[in] which FGGInputSelector register to read (1 or 2).

Return Value

On success, the value of the specified FGGInputSelector register is returned. Otherwise, the return value is 0.

Remarks

3.3.16 setNIMOutputSelector

The setNIMOutputSelector function writes a given value to the specified NIMOutputSelector register.

```
int setNIMOutputSelector {
```

int reg,
unsigned long value
};

Parameters:
Reg
[in] which NIMOutputSelector register to write (1 or 2).
value
[in] the value to write to the NIM Output Selector register.

Return Value
On Success, the return value will be >0. Otherwise the return value will be <1.

Remarks

3.3.17 getNIMOutputSelector

The getNIMOutputSelector function returns the value of the specified NIMOutputSelector register.

unsigned long getNIMOutputSelector{
int reg,
};

Parameters:
Reg
[in] which NIMOutputSelector register to read (1 or 2).

Return Value
On success, the value of the specified NIMOutputSelector register is returned. Otherwise, the return value is 0.

Remarks

3.3.18 setFGGStopSelector

The setFGGStopSelector function writes a given value to the specified FGGStopSelector register.

int setFGGStopSelector {
int reg,
unsigned long value
};

Parameters:
Reg
    [in] which FGGStopSelector register to write (1 or 2).
value
    [in] the value to write to the FGGStopSelector register.

Return Value
On Success, the return value will be >0. Otherwise the return value will be <1.

Remarks

3.3.19 getFGGStopSelector

The getFGGStopSelector function returns the value of the specified FGGStopSelector register.

unsigned long getFGGStopSelector{
    int reg,
};

Parameters:
Reg
    [in] which FGGStopSelector register to read (1 or 2).

Return Value
On success, the value of the specified FGGStopSelector register is returned. Otherwise, the return value is 0.

Remarks

3.3.20 setScalerInputSelector

The setScalerInputSelector function writes a given value to the specified ScalerInputSelector register.

int setScalerInputSelector {
    int reg,
    unsigned long value
};

Parameters:
Reg
    [in] which ScalerInputSelector register to write (1 or 2).
value
    [in] the value to write to the ScalerInputSelector register.

Return Value
On Success, the return value will be >0. Otherwise the return value will be <1.

Remarks

3.3.21 getScalerInputSelector

The getScalerInputSelector function returns the value of the specified ScalerInputSelector register.

unsigned long getScalerInputSelector{
	int reg,
};

Parameters:
Reg
	[in] which ScalerInputSelector register to read (1 or 2).

Return Value
On success, the value of the specified ScalerInputSelector register is returned.
Otherwise, the return value is 0.

Remarks

3.3.22 setLogicalMask

The setLogicalMask function writes a given value to the specified LogicalMask register.

int setLogicalMask {
	int reg,
	unsigned long value
};

Parameters:
Reg
	[in] which LogicalMask register to write (1 or 2).
value
	[in] the value to write to the LogicalMask register.

Return Value
On Success, the return value will be >0. Otherwise the return value will be <1.

Remarks

3.3.23 getLogicalMask

The getLogicalMask function returns the value of the specified LogicalMask register.

unsigned long getLogicalMask{
    int reg,
};

Parameters:
Reg
    [in] which LogicalMask register to read (1 or 2).

Return Value
On success, the value of the specified LogicalMask register is returned. Otherwise, the return value is 0.

Remarks

3.3.24 getScalerData

The getScalerData function returns the scaler value currently stored for a specified channel. The number of values currently stored for a given channel can be read via the getScalerMultiplicity function

unsigned long getScalerData{
    int channel,
};

Parameters:
channel
    [in] which which scaler to read (1-8)

Return Value
On success, the scaler value is returned. Otherwise, the return value is 0.

Remarks

3.3.25 getScalerMulitplicity

The getScalerMulitplicity function returns the depth of the scaler FIFO for a specified channel.

unsigned long getScalerMulitplicity{
    int channel,

};

Parameters:
channel

[in] which scaler multiplicity to read (1-8)

Return Value
On success, the scaler multiplicity is returned. Otherwise, the return value is 0.

Remarks

3.3.26 getCoincidenceRegister

The getCoincidenceRegister function returns the current value of the coincidence register.

unsigned long getCoincidenceRegister{
};

Parameters:

Return Value
On success, the coincidence register is returned. Otherwise, the return value is 0.

Remarks