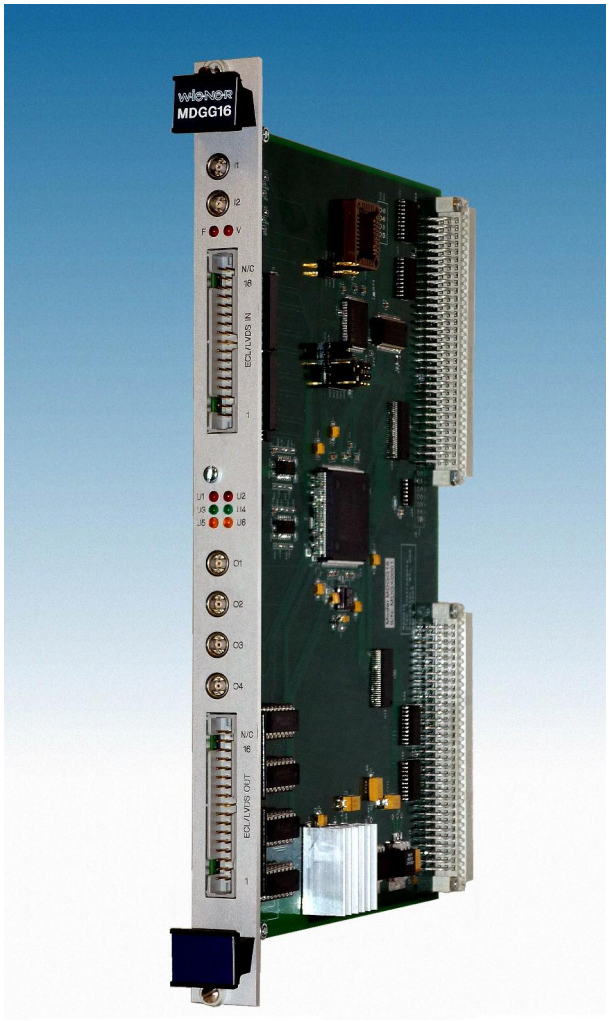
MDGG-16



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.

MDGG-8 is designed by JTEC Instruments.

Table of contents:

1 GENERAL DESCRIPTION

The MDGG-16 is a single width VME module performing multiple gate and delay generator as well as 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-16 is determined by the firmware that is loaded, the firmware capabilities must fall within the parameters of the hardware. The MDGG-16 is based on an XCS3S500E XLINX FPGA running with a 125MHz clock.
Inputs are directly both, ECL and LVDS compatible. Outputs require standard TTL->ECL translators (MC10124) for ECL-compatible outputs and custom plug-in translators for LVDS and TTL compatibility. One translator handles four outputs.
The module consists of:

INPUTS:	16 channel ECL / LVDS + NIM (LEMO 00)
OUTPUTS:	16 channel ECL / LVDS +4 NIM (LEMO 00)
LEDS:	8 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

1. Eight 32-bit flexible digital (8ns granularity) gate generators (FGG), configurable individually as
    a. digital delay and gate generators (DGG), configurable individually as
        i. non-retriggerable delay and gate generators (DGG)s,
        ii. delay + retriggerable gate generators (RDGG), or
        iii. pulse generators (PG)
    b. set-reset gates (SRG), or
    c. prescaler gates, configurable individually as

   i. 1/n prescalers (PSG) or
   ii. 1-1/n complementary prescalers (CPSG)
2. Eight standard delay and gate generators (SDGG) with fixed input and output port assignment.
3. Eight scaler devices, configurable individually as
   a. regular gated 32-bit scalers (GSC) or
   b. latchable 32-bit scalers (LSC), each with an individual 1kx32 FIFO storage
4. One 16-bit coincidence register (CREG).
5. Four combinatorial gates (CG2x8) of two-fold ORs of eightfold ANDs
6. Up-to 20-fold individual input multiplexers for flexible gates and scalers, including
   a. eight ECL inputs
   b. eight end-of-output signal (trailing edge) of the eight flexible devices, and
   c. four outputs of the four combinatorial gates.
7. 18-fold multiplexer for the scaler gating/latching signal, including
   a. eight ECL inputs,
   b. eight output signals of the eight flexible gates, and
   c. two NIM inputs.
8. 10-fold multiplexer for the scaler reset, including all eight ECL inputs and two NIM inputs.
9. Veto of trigger signals of individual flexible gates with a 10-fold selector of the common vetoing signal – any of the eight regular ECL inputs and the two control NIM inputs.
10. VME triggering and resetting of flexible gates.
11. VME reset of individual scalers.
12. Simple or block (BLT32) readout of the firmware ID, the content of the configuration registers, the eight scalers, one coincidence register, the eight latch multiplicities of the individual latched scalers (number of 32-bit words stored in individual FIFOs) and the contents of the eight latched scaler FIFOs.
13. 20-fold individual output selectors for eight ECL outputs, including
   a. eight output signals of the eight FGGs
   b. trailing edges of the eight output signals of the FGGs, and
   c. four output signals of the four combinatorial gates.
14. Four 1-of-4 multiplexed NIM outputs, each configurable to translate any one of four consecutive ECL outputs.

15. Individually selectable output polarity (invert) for all ECL (16) and NIM (4) outputs.
16. Eight diagnostic LEDs (driven by stretchers) indicating
    a. the 3-state status of the FPGA (top left red LED) :
    i. on – FPGA not configured
    ii. flashing – flash memory being programmed
    iii. off – MDGG16 configured
    b. the VME access of MDGG16 (top right red LED)
    c. the receipt of a signal at the 1st control NIM input (middle cluster, left red LED)
    d. the receipt of a signal at the 2nd control NIM input (middle cluster, right red LED)
    e. the presence of signals at ECL outputs, individual or combined (middle cluster, green and yellow LEDs). Programmable polarity.

2 VME INTERFACE

2.1 User interface

The desired configuration of MDGG16 is achieved by storing the configuration data in a set of approximately sixty registers via VME **D32 / A24** “write” commands.

2.2 Customization

MDGG16 can be readily customized within the constraints set by the hardware resources. In fact, it may be considered a 10-input/12-output universal logical module for which the user can develop custom firmware using the free XILINX WebPack software.

2.3 Firmware

The current firmware is 5A40 0100.
The firmware can be changed via VME. Jumper 25 should be in Normal / RUN position (right) for standard use. To program a new firmware please set this jumper to the PROGRAM position (left).
Please see chapter 5 for firmware upgrade instructions .

The MDGG-16 is access via the VME bus using **A24D32** read/writes or BLT reads.

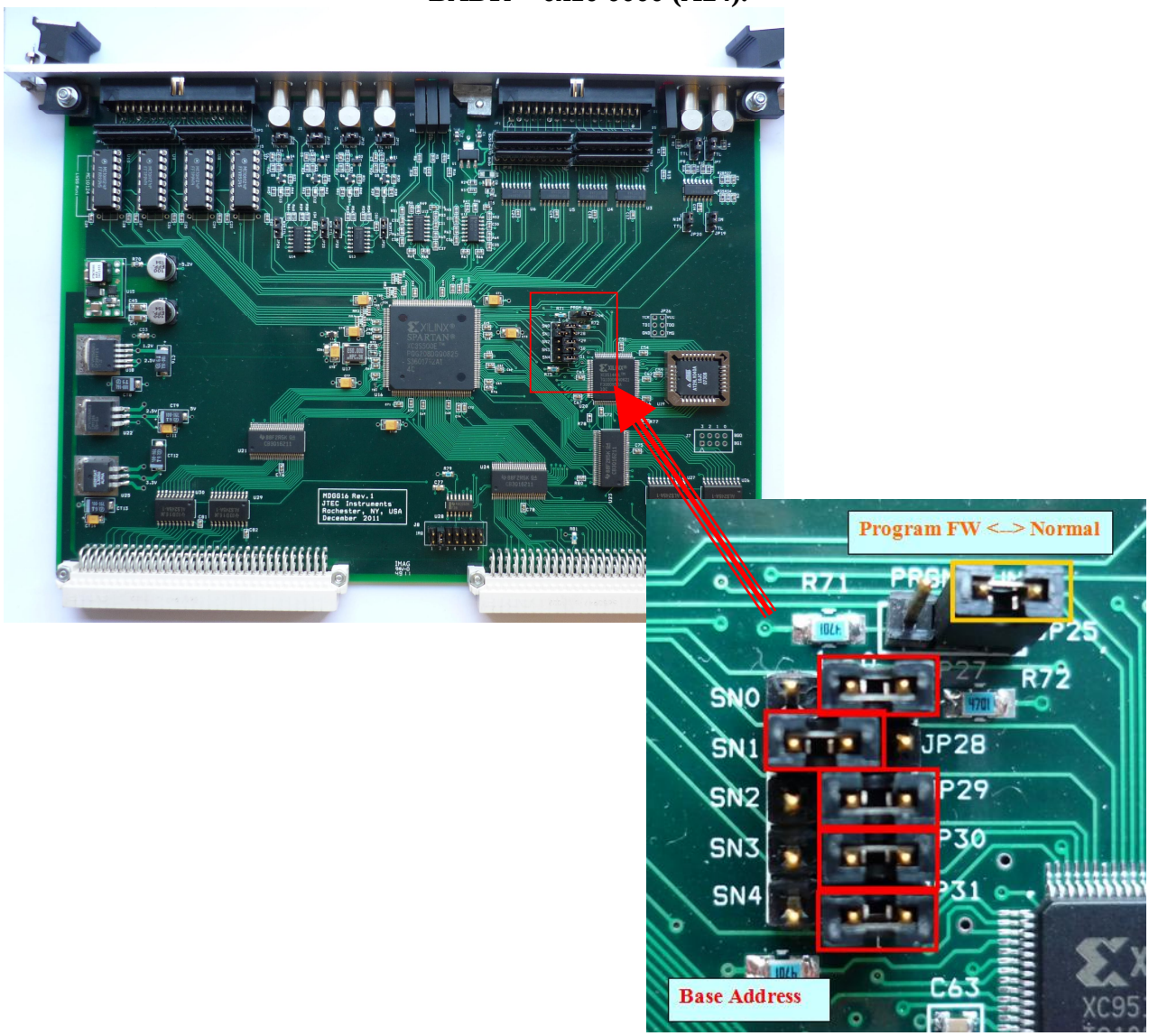
2.4 VME Base Address

The MDGG-16 base address is set via jumpers on the PCB. A jumper that is inserted in the left position (below "1") counts as a 1 in the base address A18 to A23 bit pattern.

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

The factory **default the MDGG-16 Base Address is SN1=1** (as shown in picture below) :

BADR = 0x10 0000 (A24).



2.5 VME Register Map

Offset	Register	Access Type
0x000	Firmware ID	Read only
0x004	Global Register	Read/write
0x008	Auxiliary Register	Read/write
0x010	Delay DGG9	Read/write
0x014	Gate DGG9	Read/write
0x018	Delay DGG10	Read/write
0x01C	Gate DGG10	Read/write
0x020	Delay DGG11	Read/write
0x024	Gate DGG11	Read/write
0x028	Delay DGG12	Read/write
0x02C	Gate DGG12	Read/write
0x030	Delay DGG13	Read/write
0x034	Gate DGG13	Read/write
0x038	Delay DGG14	Read/write
0x03C	Gate DGG14	Read/write
0x040	Delay FGG1	Read/write
0x044	Gate FGG1	Read/write
0x048	Delay FGG2	Read/write
0x04C	Gate FGG2	Read/write
0x050	Delay FGG3	Read/write
0x054	Gate FGG3	Read/write
0x058	Delay FGG4	Read/write
0x05C	Gate FGG4	Read/write
0x060	Delay FGG5	Read/write
0x064	Gate FGG5	Read/write
0x068	Delay FGG6	Read/write
0x06C	Gate FGG6	Read/write
0x070	Delay FGG7	Read/write
0x074	Gate FGG7	Read/write
0x078	Delay FGG8	Read/write
0x07C	Gate FGG8	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	Logical AND Mask A	Read/write
0x0B0	Logical AND Mask B	Read/write
0x0B4	IRQ	Read/write
0x0C0	Delay DGG15	Read/write
0x0C4	Gate DGG15	Read/write
0x0C8	Delay DGG16	Read/write
0x0CC	Gate DGG16	Read/write
0x0D0	LED/NIM Select	Read/write
0x100	Scaler Data 1	Read only
0x104	Scaler Data 2	Read only
0x108	Scaler Data 3	Read only
0x10C	Scaler Data 4	Read only
0x110	Scaler Data 5	Read only
0x114	Scaler Data 6	Read only
0x118	Scaler Data 7	Read only
0x11C	Scaler Data 8	Read only
0x120	Scaler Latch Multiplicity 1	Read only
0x124	Scaler Latch Multiplicity 2	Read only
0x128	Scaler Latch Multiplicity 3	Read only
0x12C	Scaler Latch Multiplicity 4	Read only
0x130	Scaler Latch Multiplicity 5	Read only
0x134	Scaler Latch Multiplicity 6	Read only
0x138	Scaler Latch Multiplicity 7	Read only
0x13C	Scaler Latch Multiplicity 8	Read only
0x140	Coincidence Register	Read only

2.5.1 Firmware ID Register (0x000)

The Firmware ID register holds the firmware revision identifier.

2.5.2 Global Register, GlobalReg (0x004)

The (32-bit) global register stores information on four global signals - common to classes of devices. These four global signals include a common veto gate for all FGGs that are configured to be veto-sensitive, a common gate signal for all gated scalers, a common latch signal for all latched scalers, and a master reset signal for FGGs, scalers, and the coincidence register. The identification of sources of global signals is achieved via 5-bit words addressing the respective signal selectors (one of twenty input signals).

Offset(GlobalReg) = 4

The structure of the global register is shown in the table below

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

Values 0 – 22 of any of the selector words represent

0	none selected
1 – 8	I1-I8, ECL inputs
9 – 16	FG1-FG8, gates generated by the 8 FGGs
17-20	CG1-CG4, combinatorial gates
21, 22	CTL, NIM inputs N1 and N2

2.5.3 Auxiliary Mask Register, AuxReg (0x008)

The 14-bit mask register stores information on the reset mask for FGGs and the coincidence register, as well as on the gate source for the latter. The mask bits determine, weather the individual FGGs or the CREG are reset by the Master Reset signal

Offset(AuxReg) = 8

The structure of the Auxiliary Mask Register is shown in the table below

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

Values 0 – 20 of any of the CReg gate selector word SelCRegGa represent

0 – 7	I1-I8, ECL inputs
8 – 15	FG1-FG8, gates generated by the 8 FGGs
16-19	CG1-CG4, combinatorial gates
21	CTL, NIM inputs N1 and N2

Note that the coincidence register must be cleared, before it is ready to accept the next gate.

2.5.4 Delay and Gate Length Registers, DelayRegFn and GateRegFn (0x0040 – 0x007C)

There are 16 32-bit registers for storing the desired values for delay and gate lengths of the eight FGGs, FGG1 – FGG8. The VME address offsets are given by the equations

Offset(DelayRegFn) = 0x38 + 8*n, for n=1-8
Offset(GateRegFn) = 0x3C + 8*n, for n=1-8.

Note that the GateReg registers are used also for storing prescale factors for prescale gates.

2.5.5 Delay and Gate Length Registers, DelayRegSn and GateRegSn (0x0010 – 0x003C, 0x00C0-0x00CC)

There are 16 32-bit registers for storing the desired values for delay and gate lengths of the eight standard delay and gate generators, SDGG1 – SDGG8. The VME address offsets are given by the equations

Offset(DelayRegSn) = 0x08 + 8*n, for n=1-6
Offset(DelayRegSn) = 0x88 + 8*n, for n=7 and 8
Offset(GateRegSn) = 0x0C + 8*n, for n=1-6
Offset(DelayRegSn) = 0x8C + 8*n, for n=7 and 8.

2.5.6 VME write-only action toggle register, ActionReg (0x0080)

The MDGG16 firmware allows one to trigger any selection of FGGs and to reset any desired selection of FGGs, scalers, or the CREG by writing 1s to the respective bits of a toggle register (bits clear automatically in 16 ns).

Offset(ActionReg) = 128 (0x80)

The structure of the action register is shown in the table below

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

Not that the actual reset signals for individual gates, scalers, and the coincidence register are three-fold logical ORs of individual VME resets, and individual and global resets derived from external sources.

2.5.7 FGG Configuration Register, FGGConfig (0x0084)

Any FGG can be configured to be either a non-retriggerable DGG, a set-reset gate, SRG, a pulser, PG, a retriggerable DGG, RDGG, a 1/n prescaler, PSG, or a complementary 1-1/n

prescaler, CPSG. Furthermore, any FGG can be configured to be subject to a common veto signal**.** The actual configuration of a single FGG is identified by a 4-bit word, with bit 3 (value 8) serving as a veto mask. Setting bit 3 = 1 sensitizes an FGG to the common veto signal.

Offset(FGGConfig) = 132 (0x84)

The structure of the FGGConfig register is shown in the table below

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

The values of the 3 least significant bits (bits0-2) of the individual configuration words represent

0 FGG off
1 DGG
2 SRG
3 PG
4 RDGG
5 PSG
6 CPSG

Bit3=1 FGG is subject to the common veto gate
Bit3=0 FGG is not subject to the common veto gate.

2.5.8 Scaler Configuration Register, SclrConfig (0x0088)

Any scaler can be configured to function either as a gated scaler, a latchable scaler, or a gated latchable scaler. In a gated scaler mode, the scaler counts trigger signals when the gate is active and suspends counting when the gate is off. In a latchable mode, scaler is active as long as the 1kx32 storage FIFO is not full and increments by 1 with every trigger signal. The state of the scaler is stored in FIFO upon the receipt of a latching signal. At the same time a latch multiplicity counter is incremented so that its content represents the number of words written into the FIFO. Both FIFO and the multiplicity counter can be read out. In the gated latchable mode, the counter is incremented only when the gate is active.

Offset(SclrConfig) = 136 (0x88)

The structure of the SCLRConfig register is shown in the table below

28-30	24-26	20-22	16-18	12-14	8-10	4-6	Bits 0-2
SCLR8	**SCLR7**	**SCLR6**	**SCLR5**	**SCLR4**	**SCLR3**	**SCLR2**	**SCLR1**

Significance of the two least significant configuration bits for an individual scaler is as follows:

"00"	scaler off
"01"	gated mode
"10"	latched mode
"11"	gated latched mode with count suspended when gating signal is off

Bit 3 of the configuration word determines whether an individual scaler responds to the Master Reset signal (1 – clears upon MReset, 0 – no action on MReset).

2.5.9 FGG Trigger Selector Registers (0x008C and 0x0090)

The trigger source for an individual FGG is identified by a 5-bit word. These words for all 8 FGGs are stored in two registers as follows:

At Offset = 0x8C

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

At Offset = 0x90

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

Values 0 – 20 of any of the FGG trigger selector word TrigSelFGGn represent

0 – 7	I1-I8, ECL inputs
8 – 15	TrEdge1-TrEdge8, trailing edges of gates generated by the 8 FGGs
16-19	CG1-CG4, combinatorial gates
20	CTL1, Top NIM Input
21	CTL2, Bottom NIM Input

2.5.10 ECL Output Source Selector Registers (0x0094 and 0x0098)

Every active Flexible Gate Generator generates two signals, the gate and its trailing edge marker (TrEdge). Further, every combinatorial gate generates one signal representing the result of the logical OR of 8-fold logical ANDs of selected input signals. Each of the above

signals can be routed to any of the ECL outputs (O1-O8) of MDGG16, by writing a proper 5-bit code into one of two ECL Output Source Selector Registers. The structure of these registers is as follows:

At Offset = 0x94

Bits24-28	Bits 16 – 20	Bits 8 – 12	Bits 0 - 4
ECLOutSel 4	**ECLOutSel 3**	**ECLOutSel 2**	**ECLOutSel1**

At Offset = 0x98

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

Values 0 – 19 of any of the output source selector word **NIMOutSel** n represent

0 – 7 Gate1-Gate8, gate signals of the 8 FGGs
8 – 15 TrEdge1-TrEdge8, trailing edges of gates generated by the 8 FGGs
16-19 CG1-CG4, combinatorial gates

2.5.11 Gate Reset Signal Selector Registers (0x009C and 0x00A0)

Any FGG configured as a Set-Reset latch is triggered (set) by a trigger signal as defined by it 5-bit code in the Trigger Source Selector Register. It is reset either by a global reset signal (subject to FGG Reset Mask), by its individual VME reset signal, or by a signal identified by a 5-bit code in one of the two Gate Reset Signal Selector Registers. The latter registers have the following structure:

At Offset = 0x9C

Bits24-28	Bits 16 – 20	Bits 8 – 12	Bits 0 - 4
StopSelFGG4	**StopSelFGG3**	**StopSelFGG2**	**StopSelFGG1**

At Offset = 0xA0

Bits24-28	Bits 16 – 20	Bits 8 – 12	Bits 0 - 4
StopSelFGG8	**StopSelFGG7**	**StopSelFGG6**	**StopSelFGG 5**

Values 0 – 21 of any of the FGG tstop selector words **StopSelFGG**n represent

0 – 7 I1-I8, ECL inputs

8 – 15	TrEdge1-TrEdge8, trailing edges of gates generated by the 8 FGGs
16-19	CG1-CG4, combinatorial gates
20, 21	CTL, NIM inputs, N1 and N2

2.5.12 Scaler Input Selector Register (0x00A4 and 0x00A8)

Input source for an individual scaler is identified by a 5-bit word. These words for all 8 scalers are stored in two registers as follows:

At Offset = 0xA4

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

At Offset = 0xA8

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

Values 0 – 21 of any of the scaler input selector word SclrInSeln represent

0 – 7	I1-I8, ECL inputs
8 – 15	TrEdge1-TrEdge8, trailing edges of gates generated by the 8 FGGs
16-19	CG1-CG4, combinatorial gates
20, 21	CTL, NIM inputs N1 and N2

2.5.13 Diagnostic LED and NIM Output Setup Register (0x00D0)

MDGG16 is equipped with 8 diagnostic LEDs, four of which are of configurable functionality. It is also equipped with 4 NIM outputs that are configurable in that they can be associated with individual ECL outputs. The functionality of LEDs and NIM outputs is determined by the content of a setup register at offset 0xD0 as follows:

Bits 28-31	Bits 24-27	Bits 20-23	Bits 16-19	Bits 12-15	Bits 8-11	Bits 4-7	Bits 0-3
NIM 4	NIM 3	NIM 2	NIM 1	LED YR	LED YL	LED GR	LED GL

Values 0 – 7 of LED source selector represent:

0 – quadruple OR of ECL output signals 1-4 (GL), 5-8 (GR), 9-12 (YL), 13-16 (YR)

1 – double OR of ECL output signals 1 and 2 (GL), 5 and 6 (GR), 9 and 10 (YL), 13 and 14 (YR)

2 – double OR of ECL output signals 2 and 3 (GL), 6 and 7 (GR), 10 and 11 (YL), 14 and 15 (YR)

3 – double OR of ECL output signals 3 and 4 (GL), 7 and 8 (GR), 11 and 12 (YL), 15 and 16 (YR)

4 – ECL output signal 1 (GL), 5 (GR), 9 (YL), 13 (YR)

5 – ECL output signal 2 (GL), 6 (GR), 10 (YL), 14 (YR)

6 – ECL output signal 3 (GL), 7 (GR), 11 (YL), 15 (YR)

7 – ECL output signal 4 (GL), 8 (GR), 12 (YL), 16 (YR)

GL, GR, YL, and YR refer to left and right green and yellow LEDs.

Values 0 – 5 of NIM output source selector word represent:

0, 5, 6, and 7 – none

1 – ECL output 1 (NIM 1), 5 (NIM 2), 9 (NIM3), 13 (NIM4)

2 – ECL output 2 (NIM 1), 6 (NIM 2), 10 (NIM3), 14 (NIM4)

3 – ECL output 3 (NIM 1), 7 (NIM 2), 11 (NIM3), 15 (NIM4)

4 – ECL output 4 (NIM 1), 8 (NIM 2), 12 (NIM3), 16 (NIM4)

Bit 3 of every individual 4-bit selector word determines the polarity of the signal, such that setting of bit 3 causes inverting of the signal.

2.5.14 Combinatorial Gates Mask Registers (0x00AC and 0x00B0)

Each of the four combinatorial gates of MDGG16 represent a two-fold logical OR of (up-to 8-fold) logical ANDs of selected NIMn (n=1-8) inputs. The selection of active NIM inputs is achieved via 8-bit AND masks (AMASK(1:4,1:2)), such that the active inputs have their respective bits set to 1, while inactive inputs have bits reset to 0. Thus, the logical equation for the n-th combinatorial gate CGn reads:

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

where NIM is an 8-bit word representing the status of the 8 NIM inputs.

Note that any inactive term in the above equation must have all mask bits set to 1 and not to 0. Obviously, in the latter case, the result would be true for no NIM inputs present.

The 8 AMASK words are stored in two registers in the following manner

At Offset = 0xAC

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

At Offset = 0xB0

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

2.5.15 Scaler Data Registers (0x0100 – 0x013C)

MDGG16 stores scaler data in respective 1kx32 FIFO's, for read-back via VME commands. The number of words in FIFO's is stored in respective hit multiplicity registers. For scalers operated in gated mode, only one word is stored in the data FIFO with the hit multiplicity being always 1. A gated scaler must be reset, before it is ready to accept subsequent gate. For latched scalers, FIFO's contain the latched states of the counters. A latched scalers is active as long as its associated FIFO is not full, subject to the gating signal, if applicable (gated latched mode). VME address offsets for the Scaler FIFO's and Hit Multiplicities are as follows:

Offset(FIFOn) = 0xFC+n*4, n=1-8
Offset(HitMultn) = 0x11C + n*4, n=1-8

2.5.16 Coincidence Register Data Word (0x0140)

The state of the coincidence register CReg is available for readout at VME offset 0x140. Note that the register must be cleared, before it is ready to accept a new gate.

2.5.17 IRQ Register (0x00B4)

The Interrupt register IRQReg is is used for interrupt definitions

Bits24-31	Bits 16 – 23	Bits 8 – 12	Bits 0 – 2
AMASK(2,2)	AMASK(2,1)	Trigger	IRQ level

IRQ level: value 1-7, 0-disabled
Trigger: trigger selector, 1 out of 22 (same mux input as for scalers)

VETO: -20 - veto selector - 1 out of 22 (same as above)
>> bits 24-31 - IRQ ID
>> . Note that the register must be cleared, before it is ready to accept a new gate.

3 REGISTER COMPARISON MDGG-8 VS. MDGG-16

Offset	MDGG-8 Register	MDGG-16 Register	Access Type
0x000	Firmware ID	Firmware ID	Read only
0x004	Global Register	Global Register	Read/write
0x008	Auxiliary Register	Auxiliary Register	Read/write
0x010		Delay DGG9	Read/write
0x014		Gate DGG9	Read/write
0x018		Delay DGG10	Read/write
0x01C		Gate DGG10	Read/write
0x020		Delay DGG11	Read/write
0x024		Gate DGG11	Read/write
0x028		Delay DGG12	Read/write
0x02C		Gate DGG12	Read/write
0x030		Delay DGG13	Read/write
0x034		Gate DGG13	Read/write
0x038		Delay DGG14	Read/write
0x03C		Gate DGG14	Read/write
0x040	Delay FGG1	Delay FGG1	Read/write
0x044	Gate FGG1	Gate FGG1	Read/write
0x048	Delay FGG2	Delay FGG2	Read/write
0x04C	Gate FGG2	Gate FGG2	Read/write
0x050	Delay FGG3	Delay FGG3	Read/write
0x054	Gate FGG3	Gate FGG3	Read/write
0x058	Delay FGG4	Delay FGG4	Read/write
0x05C	Gate FGG4	Gate FGG4	Read/write
0x060	Delay FGG5	Delay FGG5	Read/write
0x064	Gate FGG5	Gate FGG5	Read/write
0x068	Delay FGG6	Delay FGG6	Read/write
0x06C	Gate FGG6	Gate FGG6	Read/write
0x070	Delay FGG7	Delay FGG7	Read/write
0x074	Gate FGG7	Gate FGG7	Read/write
0x078	Delay FGG8	Delay FGG8	Read/write
0x07C	Gate FGG8	Gate FGG8	Read/write
0x080	Action Register	Action Register	Write only
0x084	FGG Configuration	FGG Configuration	Read/write
0x088	Scaler Configuration	Scaler Configuration	Read/write
0x08C	FGG Input Selector A	FGG Input Selector A	Read/write
0x090	FGG Input Selector B	FGG Input Selector B	Read/write
0x094	NIM Output Selector A	NIM Output Selector A	Read/write
0x098	NIM Output Selector B	NIM Output Selector B	Read/write
0x09C	FGG Stop Selector A	FGG Stop Selector A	Read/write
0x0A0	FGG Stop Selector B	FGG Stop Selector B	Read/write
0x0A4	Scaler Input Selector A	Scaler Input Selector A	Read/write
0x0A8	Scaler Input Selector B	Scaler Input Selector B	Read/write
0x0AC	Combination Gate Mask A	Logical AND Mask A	Read/write
0x0B0	Combination Gate Mask B	Logical AND Mask B	Read/write
0x0B4		IRQ	Read/write

0x0C0		Delay DGG15	Read/write
0x0C4		Gate DGG15	Read/write
0x0C8		Delay DGG16	Read/write
0x0CC		Gate DGG16	Read/write
0x0D0		LED/NIM Select	Read/write
0x100	Scaler Data 1	Scaler Data 1	Read only
0x104	Scaler Data 2	Scaler Data 2	Read only
0x108	Scaler Data 3	Scaler Data 3	Read only
0x10C	Scaler Data 4	Scaler Data 4	Read only
0x110	Scaler Data 5	Scaler Data 5	Read only
0x114	Scaler Data 6	Scaler Data 6	Read only
0x118	Scaler Data 7	Scaler Data 7	Read only
0x11C	Scaler Data 8	Scaler Data 8	Read only
0x120	Scaler Latch Multiplicity 1	Scaler Latch Multiplicity 1	Read only
0x124	Scaler Latch Multiplicity 2	Scaler Latch Multiplicity 2	Read only
0x128	Scaler Latch Multiplicity 3	Scaler Latch Multiplicity 3	Read only
0x12C	Scaler Latch Multiplicity 4	Scaler Latch Multiplicity 4	Read only
0x130	Scaler Latch Multiplicity 5	Scaler Latch Multiplicity 5	Read only
0x134	Scaler Latch Multiplicity 6	Scaler Latch Multiplicity 6	Read only
0x138	Scaler Latch Multiplicity 7	Scaler Latch Multiplicity 7	Read only
0x13C	Scaler Latch Multiplicity 8	Scaler Latch Multiplicity 8	Read only
0x140	Coincidence Register	Coincidence Register	Read only

4 SOFTWARE SUPPORT

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

4.1 Microsoft Windows Support

WIENER support of the MDGG-8/16 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 / MDGG-16 and VM-USB. Additionally, a prepackaged executable is available for setting up and reading the MDGG-8 / MDGG-16. This executable is an extension of the XXUSBwin program shipped with the VM-USB.

4.2 Linux Support

WIENER supports Linux users by providing an open source C++ library to allow for easy setup and readout of the MDGG-8/16 when used in combination with the WIENER VM-USB.

4.3 MDGG 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-8-16 class for performing all of the necessary tasks for setting up and reading an MDGG-8/16 module.

4.3.1 CMDGG

The CMDGG is the constructor of the MDGG-8-16 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

4.3.2 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

4.3.3 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

4.3.4 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

4.3.5 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 or 9 -16 for DGG gates)
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

4.3.6 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.

4.3.7 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 or 9 -16 for DGG gates)

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

4.3.8 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.

4.3.9 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.

4.3.10 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

4.3.11 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

4.3.12 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

4.3.13 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

4.3.14 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

4.3.15 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

4.3.16 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

4.3.17 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

4.3.18 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

4.3.19 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

4.3.20 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

4.3.21 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

4.3.22 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

4.3.23 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

4.3.24 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

4.3.25 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

4.3.26 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

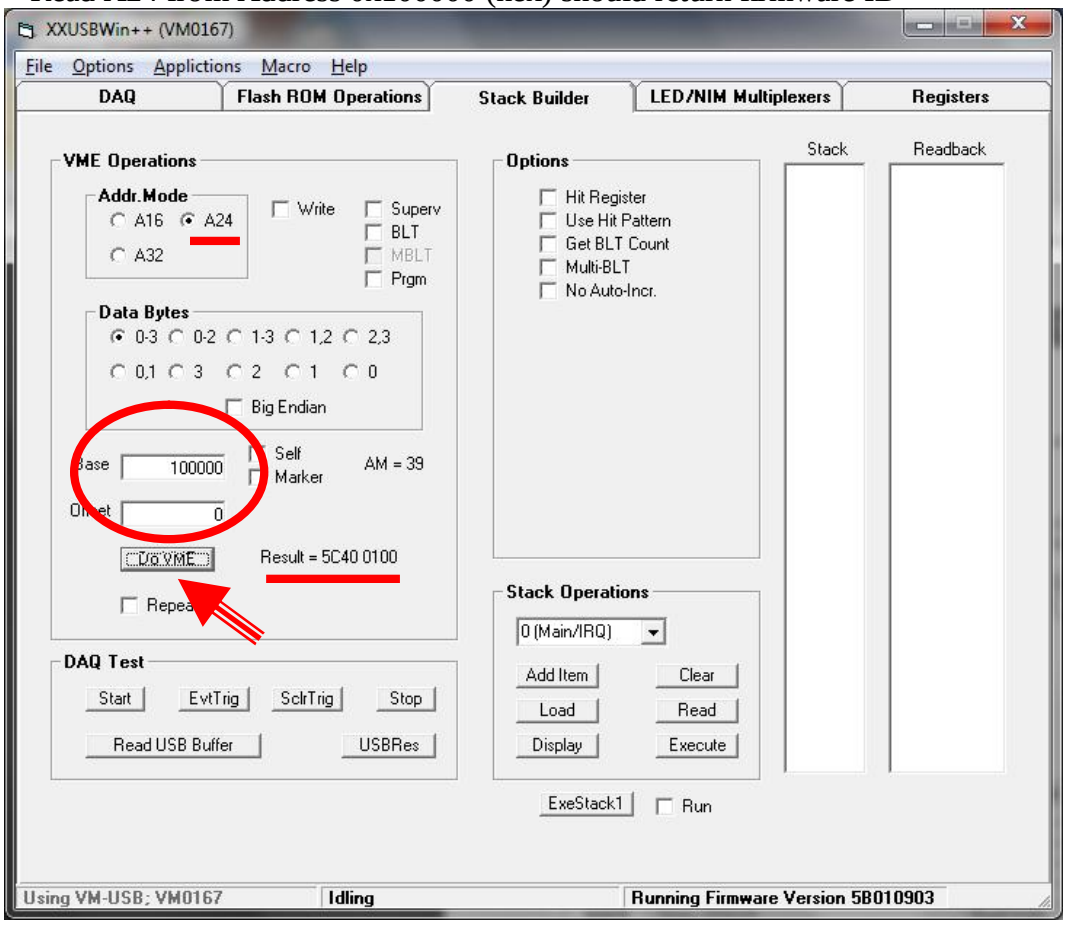
5 FIRMWARE UPGRADE PROCEDURE

Hardware needed: VM-USB
Software: XXUSBWinAppli.exe

1. Check Base address of MDGG-16 (following example shown for factory default bade address 0x100000)

2. Check current Firmware Version with XXUSBWinAppli.exe:

Leave Jumper JP 25 in right (RUN) position, Power VME crate,
Run XXUSBWinAppli.exe
Read A24 from Address 0x100000 (hex) should return firmware ID

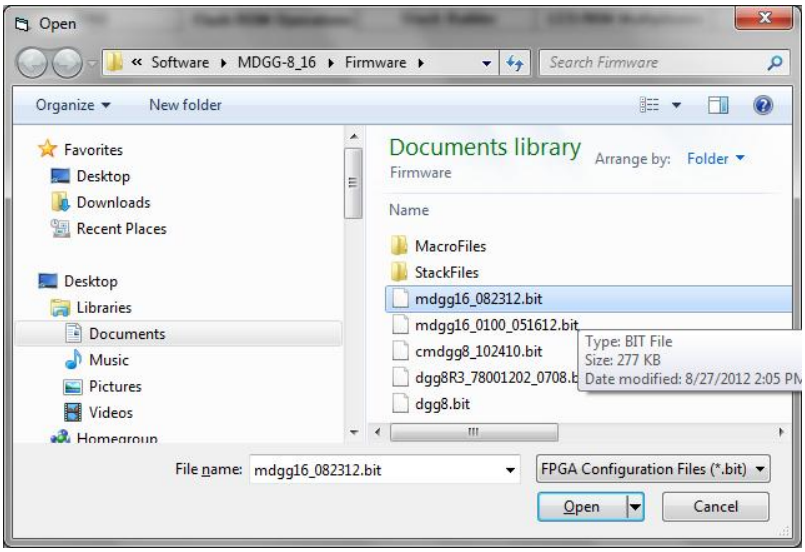


➔ Response: 5C40 0100

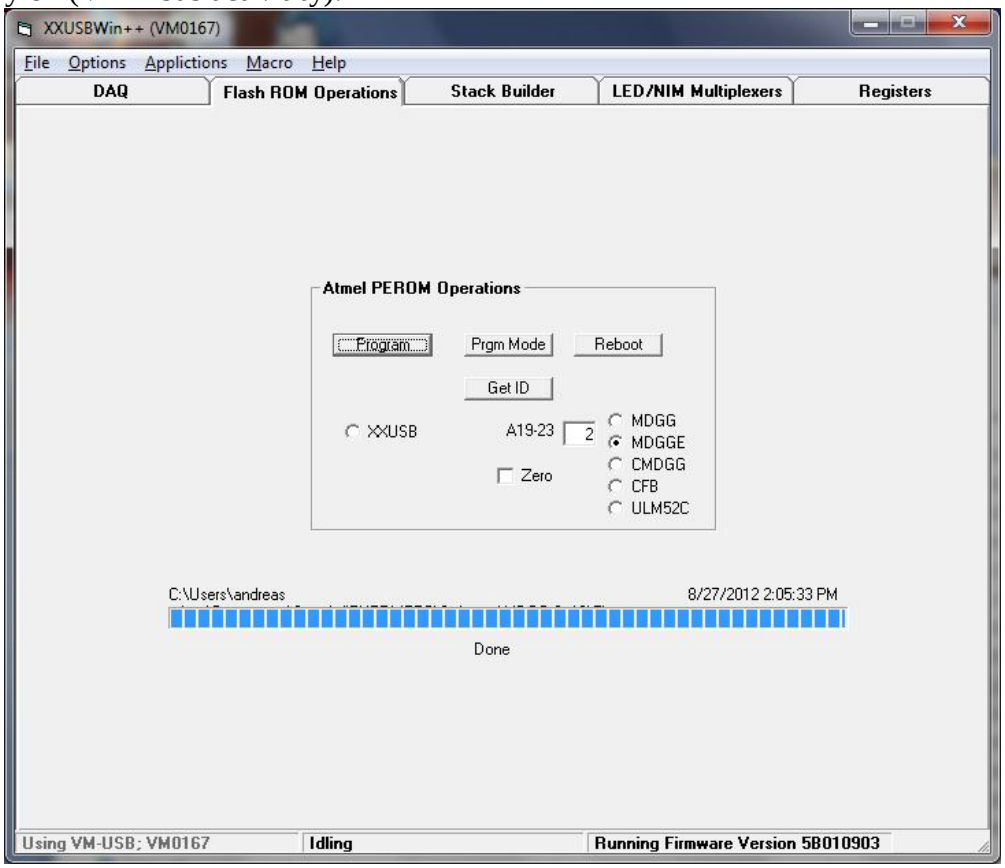
3. Program new Firmware

Set Jumper 25 into left (Programm Position)
Run XXUSBWinAppli.exe
Start Go to “Flash ROM Operations” tab
Select MDGGE

Type in value for Base Address (bits 19 to 23), is **2 for factory default**!!!
Click on program and select the right firmware file



When programming the red F LED will flash and the red V LED will be flashing very fast / be nearly on (VME bus activity).



4. Check new Firmware Version with XXUSBWinAppli.exe:

Return Jumper JP 25 into right (RUN) position, Power VME crate,
Run XXUSBWinAppli.exe
Read A24 from Address 100000 (hex) should return firmware ID
Compare with original version