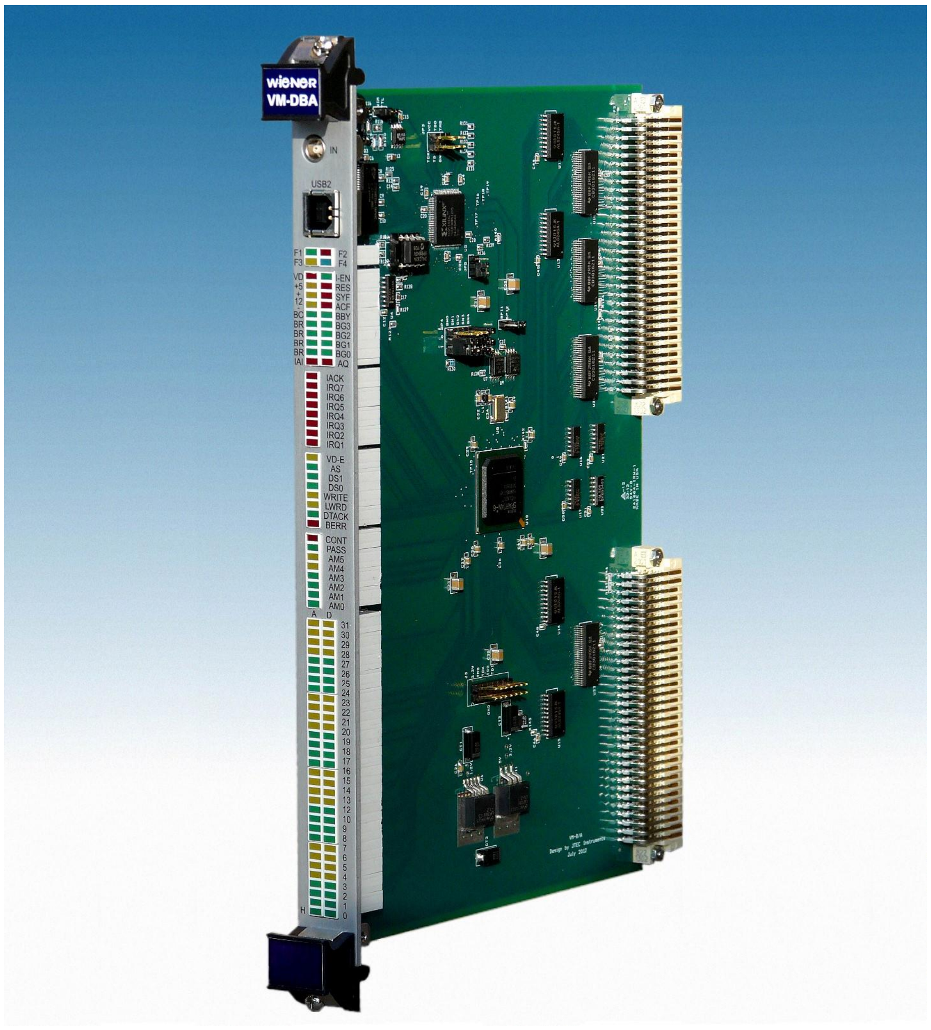
VM-DBA



User Manual

Version 0.0

General Remarks

VM-DBA is designed by JTEC Instruments.

Table of contents:

1 GENERAL DESCRIPTION

The VM-DBA is a new test and diagnostic module for VME / VME64x bus systems which visualizes activities on all VME bus lines on the set of front-panel LED's, but also allows one to digitize individual waveforms of all these bus lines into 2kBit long storage memories, for a subsequent readout via VME or USB interfaces.

The digitization of waveforms is triggered by a programmable selection of conditions, including an external signal received at the front-panel LEMO connector. Pre-and post trigger sample ranges can be defined.

1.1 VM-DBA features

The VM-DBA provides all functionality of the predecessor VDIS-2 but adds a full VME-bus analyzer with VME and USB read-out as well as additional test and diagostic functions.

- Single-width 6U VME / VME64x slave module
- VME-bus-Slave D32 and A16, A24, A32. Interrupt handler 1H(1-7)
- conform to VME-bus ANSI/IEEE STD 1014, IEC821 and IEC297
- Dual user interface - VME and USB
- LED indicators for all VME bus lines:
- 32 data and 32 address lines
- interrupt lines (IRQ1-7 , IACK, IACKIN)
- Bus Clear, Bus Busy, BG1 - BG3 and BR1 - BR3
- control signals (VD, CLK, RES, SYSF, ACF, AS, DS1, DS0, LWORD, DTACK, BERR)
- address modifier (AM0 - AM5)
- supply voltages (+5V, +/-12V, +3.4V)
- 4 user programmable front panel LED's, status and 5 control LED's
- 32-bit read and write register, accessed if no board answers on the VME-bus after a suitable time-out and asserts DTACK
- Built-in full functionality of VDIS-2 predecessor module:
- Two on-board, user-programmable SPI memories to store 2 different FPGA configurations.
- Built in VME bus analyzer with 2k memory for all bus lines, sampling with 100MHz or 200MHz, user- programmable trigger for waveform digitization and pre- / post trigger time range
- NIM / TTL input (jumper selectable)
- Selection of user-programmable registers and memories for comprehensive testing of VME bus operations, including IRQ handling and bus arbitration.
- Functionality based on a Xilinx XC6SLX100 FPGA
- Low power CMOS-Technology: power consumption <0.8 A

2 INTERFACE / FIRMWARE INFORMATION

2.1 VME chassis / bus slot location

Several VME bus lines as for instance interrupts or bus grant requests are daisy chained on the VME backplane. In order for displaying and diagnosing these correctly the VM-DBA should be placed between the VME master (typically in system slot 1 on the left) and the first slave module right of it.. Thus a suggested slot position would be slot 2. The VM-DBA should not be located left of a VME master or to the right of any other slave module.

2.2 User interface

The VM-DBA bus display and bus analyzer is D32 and A16/A24/A32 VME slave module with additional USB-2 interface for communication and data transfer.

2.3 Firmware

VM-DBA has two on-board, user-programmable SPI memories to store 2 different FPGA configurations. SPI memory location 2 is the active one whereas location 1 is defined as protected and can be used to restore operation and reprogram the active firmware in case of a failure. Jumper 6 selects the active SPI for boot and should be kept in SPI2 position.

The current firmware is 80D00002.

The firmware can be updated via VME and USB. Please see chapter 10 for firmware upgrade instructions.

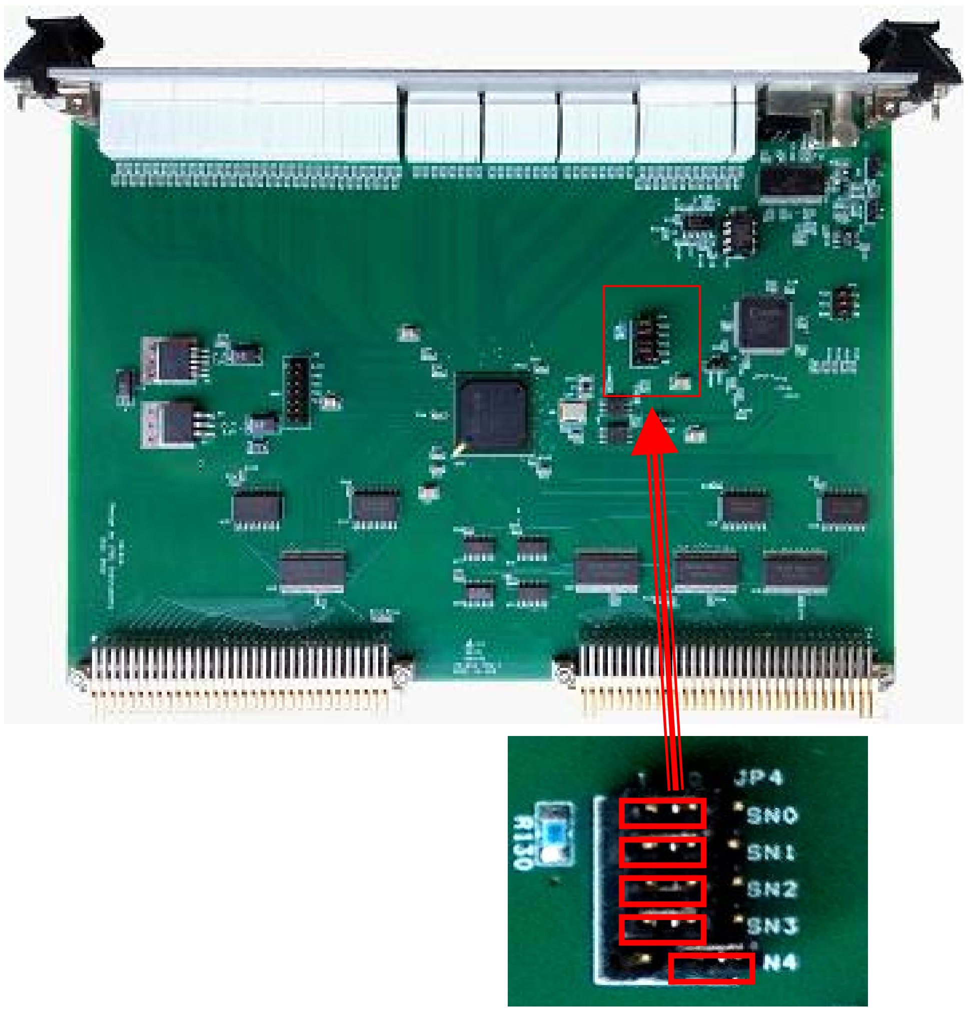
2.4 VME Base Address

The VM-DBA base addresses for all modes (A16/A24/A32) are set via jumpers in the JP 4 array on the PCB. A jumper that is inserted in the left position (below “1”) counts as a 1 in the base address bit pattern.

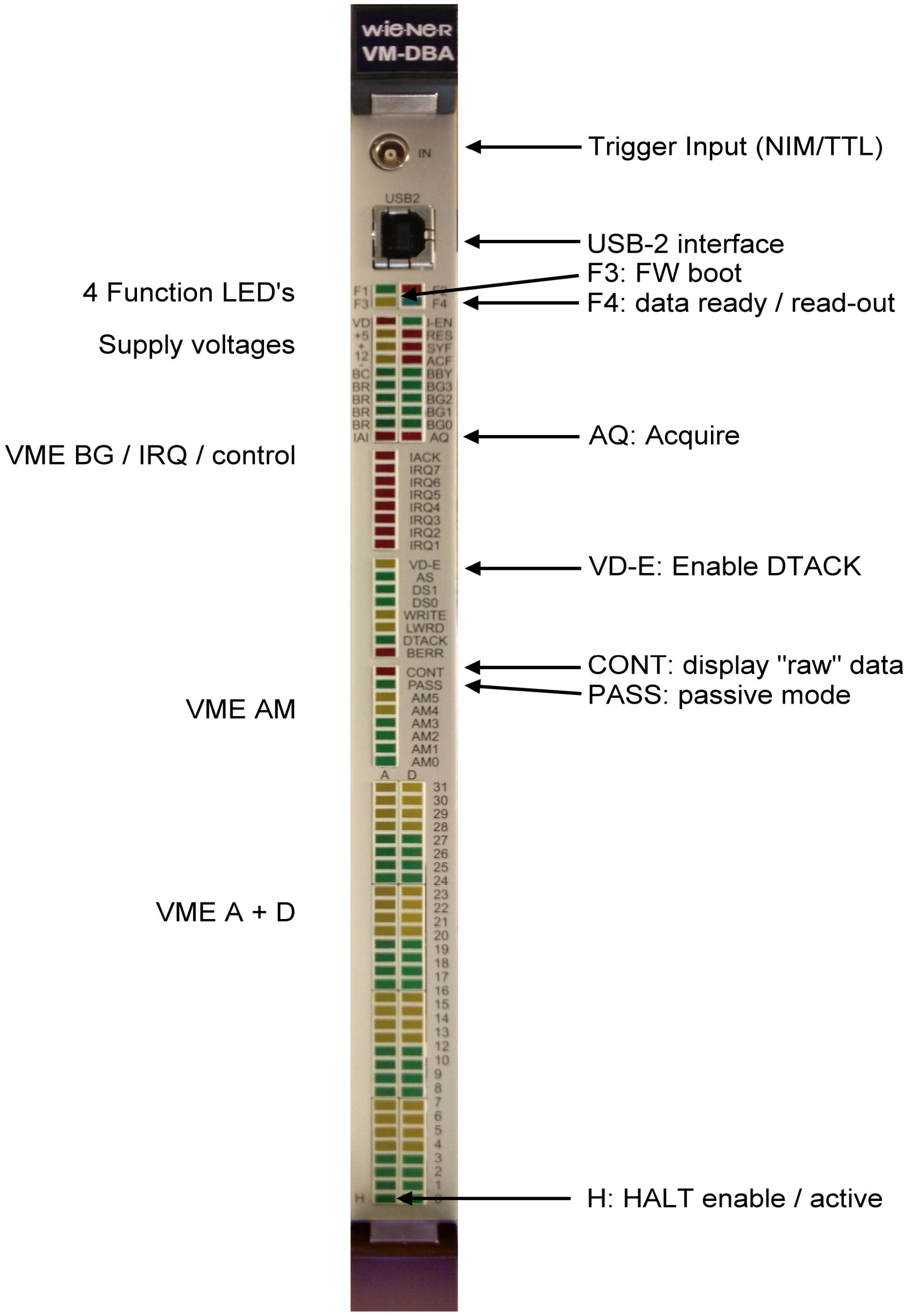
Mode	**SN4**	**SN3**	**SN2**	**SN1**	**SN0**	**-**	**Default BADR**
A16	A15	A14	A13	A12	A11	A10…0	**0x7800**
A24	A23	A22	A21	A20	A19	A18…0	**0x78 0000**
A32	A31	A30	A29	A28	A27	A26…0	**0x7800 0000**
Jumper	**0**	**1**	**1**	**1**	**1**	**-**	

The factory **default the VM-DBA Base Address is SN0=SN1=SN2=SN3=1 and** as shown in picture below.

BADR16 = 0x7800 (A16).
BADR24 = 0x78 0000 (A24).
BADR32 = 0x7800 0000 (A32).



3 VM-DBA LED DISPLAY



VM-DBA visualizes activity on the VME bus lines in three different ways, direct, latched, and stretched. In the direct or “raw” mode, the LED’s indicate the actual state of the lines, while in latched mode, the states of the lines are captured either by the DTACK*, or by BERR*, or by BBSY*, or by a pseudo-DTACK* signal generated by VM-DBA itself. In the extended mode, the states of the signals are stretched so as to be at least 15 ms long, i.e., long enough to be visible on the display in a form of a blink.

The following VME bus signals are captured by DTACK* or BERR*:

• Data lines:	D0 – D31
• Address lines:	A1 – A31
• Address modifier lines:	AM0 – AM5
• Control signals:	AS*, DS0*, DS1*, WRITE*, LWORD*, DTACK*,BERR*
• Interrupt Bus:	IACK*, IACKIN*

When in a pseudo-DTACK mode, VM-DBA generates itself a DTACK* upon determining the absence of DTACK* signal within a timeout period beginning at AS*. This pseudo-DTACK latches also the above bus signals.

The following signals are captured by BBSY*:

• Arbitration bus:	BGIN0* – BGIN7*

The following signals are made visible by stretching:

• Interrupt bus:	IRQ1* - IRQ7*
• Arbitration bus:	BR0* - BR3*, BCLR*, BBSY*
• Utility bus:	SYSRESET*, SYSFAIL*, ACFAIL*

The state of power lines +5V, +3.3V, +12V, and -12V and the SYSCLK line are displayed "as-is".

In addition to displaying the state of the VME bus lines, VM-DBA displays on selected LED's its operating mode or state, as well as the occurrence of events of importance, as indicted in the table below.

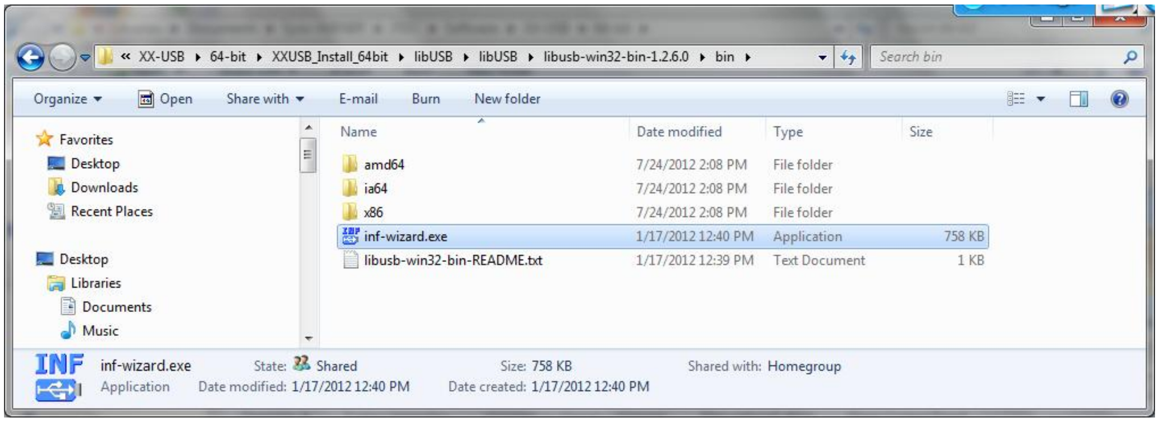
Row/Col	Color	Label	Function
1L	**G**	**F1**	"FPGA Fail" or "Booting", or "Accessed", or "IN", or "IRQ"
1R	**R**	**F2**	Sampling rate – "on" indicates 200 MS, "off" is for 100 MS
2L	**Y**	**F3**	Active memory –"on" for SPI-1, "off" for the (protected) SPI-0
2R	**B**	**F4**	Status of the waveform acquisition – blinking when waveforms stored, steady "on" when waveforms successfully read out.
3L	**R**	**VD**	pseudo DTACK generated (latched)
3R	**G**	**I-EN**	Interrupt generation and handling active
12R	**R**	**AQ**	Continuous waveform writing into circular FIFOs
21	**Y**	**VD-E**	Pseudo-DTACK generation enabled
29	**R**	**CONT**	Display "raw", as-is VME bus line states
30	**G**	**PASS**	Passive mode
68L	**G**	**H**	Blinking when Halt-enabled, steady "on" when display halted

4 VM-DBA SOFTWARE AND DRIVER

4.1 VM-DBA USB driver installation on 32 and 64-bit MS Windows Systems

DRIVER INSTALLATION

1. Connect VM-DBA to the computer via USB and power up the VME crate.
2. Run inf-wizard.exe from libusb-win32-bin-1.2.6.0 -> this will identify all USB devices and should find the WIENER VM-DBA among them

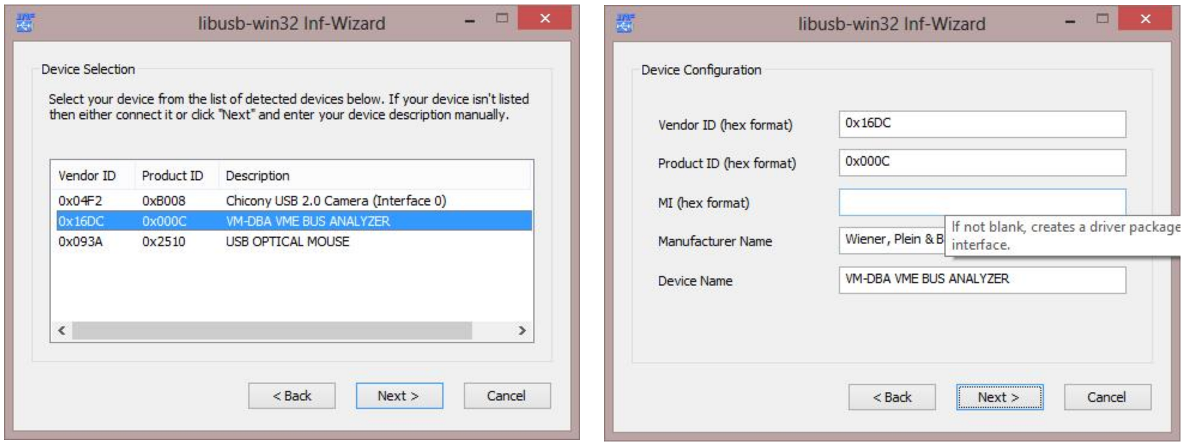


« XX-USB ▸ 64-bit ▸ XXUSB_Install_64bit ▸ libUSB ▸ libUSB ▸ libusb-win32-bin-1.2.6.0 ▸ bin ▸

Name	Date modified	Type	Size
amd64	7/24/2012 2:08 PM	File folder	
ia64	7/24/2012 2:08 PM	File folder	
x86	7/24/2012 2:08 PM	File folder	
inf-wizard.exe	1/17/2012 12:40 PM	Application	758 KB
libusb-win32-bin-README.txt	1/17/2012 12:39 PM	Text Document	1 KB

inf-wizard.exe State: Shared Size: 758 KB Shared with: Homegroup
Application Date modified: 1/17/2012 12:40 PM Date created: 1/17/2012 12:40 PM

3. Select the VM-DBA and generate + save the VM-DBA_VME_BUS_ANALYZER.inf file to disk.



libusb-win32 Inf-Wizard

Device Selection

Select your device from the list of detected devices below. If your device isn't listed then either connect it or click "Next" and enter your device description manually.

Vendor ID	Product ID	Description
0x04F2	0xB008	Chicony USB 2.0 Camera (Interface 0)
0x16DC	0x000C	VM-DBA VME BUS ANALYZER
0x093A	0x2510	USB OPTICAL MOUSE

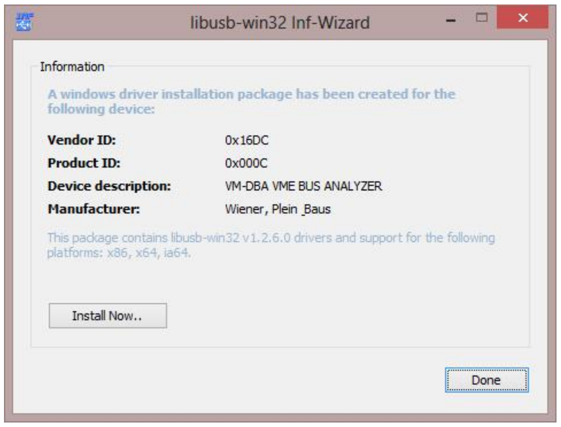
libusb-win32 Inf-Wizard

Device Configuration

Vendor ID (hex format)	0x16DC
Product ID (hex format)	0x000C
MI (hex format)	
Manufacturer Name	Wiener, Plein & B
Device Name	VM-DBA VME BUS ANALYZER

If not blank, creates a driver package interface.

3. Select " Install Now …", this will install the driver files into proper directories.


libusb-win32 Inf-Wizard

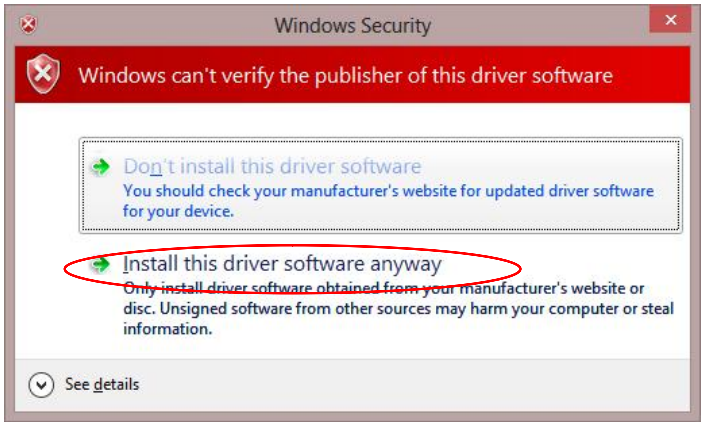
Information

A windows driver installation package has been created for the following device:

Vendor ID:	0x16DC
Product ID:	0x000C
Device description:	VM-DBA VME BUS ANALYZER
Manufacturer:	Wiener, Plein _Baus

This package contains libusb-win32 v1.2.6.0 drivers and support for the following platforms: x86, x64, ia64.

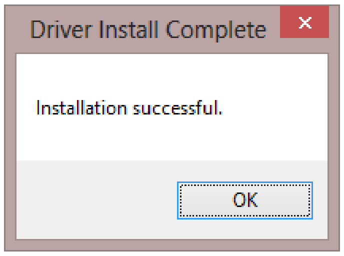
4. Ignore warning for unknown publisher


Windows Security

Windows can't verify the publisher of this driver software

Don't install this driver software
You should check your manufacturer's website for updated driver software for your device.

Install this driver software anyway
Only install driver software obtained from your manufacturer's website or disc. Unsigned software from other sources may harm your computer or steal information.

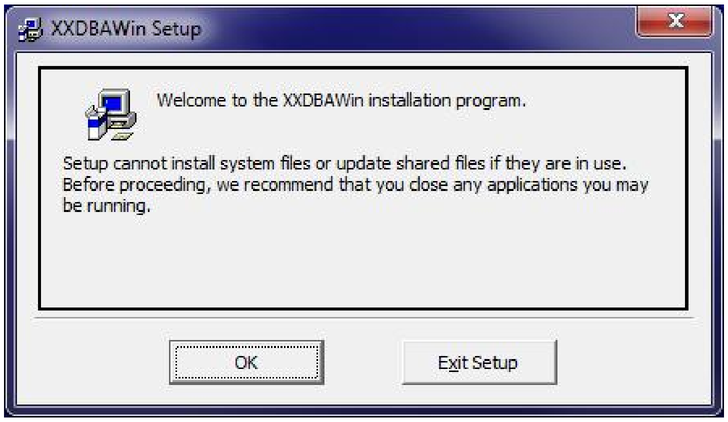

Driver Install Complete

Installation successful.

4.2 XXDBAWin Software

4.2.1 XXDBAWin software installation

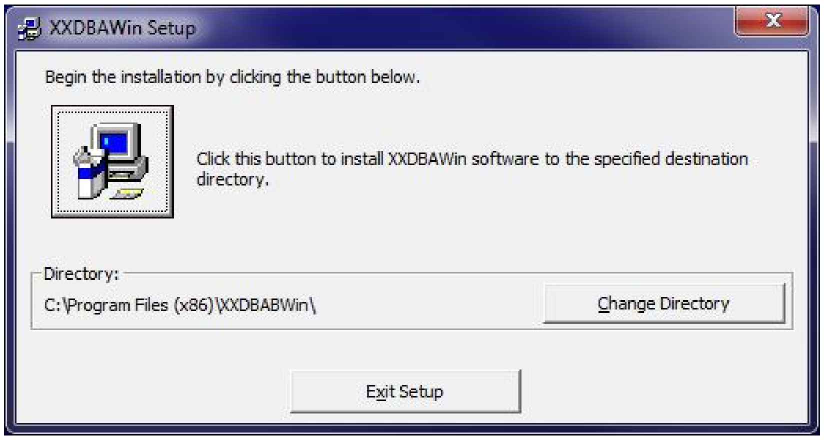
To install the software run SETUP.EXE in the CD-ROM xxdbawin_install directory:


XXDBAWin Setup

Welcome to the XXDBAWin installation program.

Setup cannot install system files or update shared files if they are in use. Before proceeding, we recommend that you close any applications you may be running.

Change the destination folder in case needed and / or click on the computer button to start the installation process:


XXDBAWin Setup

Begin the installation by clicking the button below.

Click this button to install XXDBAWin software to the specified destination directory.

Directory:
C:\Program Files (x86)\XXDBABWin\

After installation the program group / program XXDBAWin should be added to the programs which can be used to start the program.

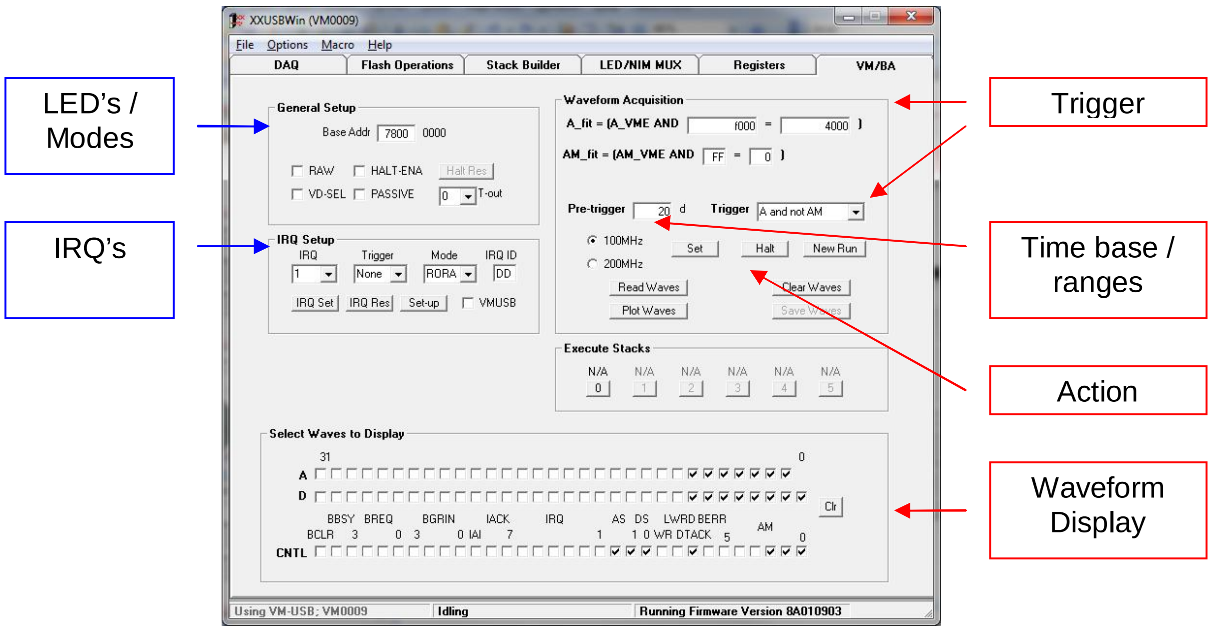
To un-install the program use the Windows Control Panel >> Programs >> Programs and Features.

The XXDBAWin software allows to control the VM-DBA through both the on-board USB interface as well as through the VME bus when using the WIENER VM-USB controller as the VME master. Please refer to the VM-USB manual for programming and use of this controller.

In case of VM-DBA operation through it’s USB port only the “VM/DBA” tab for setting and reading data as well as the “Flash Operations” tab for firmware upgrades are used. All other tabs are disabled. Please see chapter 7 for firmware upgrade instructions.

Using the XXDBAWin software it is possible to configure the display modes / options (General setup) and define the VM-DBA interrupt settings (IRQ setup).

The VME bus analyzer can be configured by defining the trigger conditions as well as time base and ranges. When done the trigger can be activated and it is possible to read back data when available. XXDBAWin provides a graphic display of the bus lines as a function of time. All bus lines to be shown on the waveform display can be selected and the waveform histogram is shown automatically.



Base Addr 7800 0000

RAW [ ] HALT-ENA [ ] VD-SEL [ ] PASSIVE [ ] 0 T-out

A_fit = (A_VME AND f000 = 4000)

AM_fit = (AM_VME AND FF = 0)

Pre-trigger 20 d Trigger A and not AM

[✓ 100MHz] 200MHz

IRQ 1 Trigger None Mode RORA IRQ ID DD VMUSB [ ]

```
    31                                                                              0
A   □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ☑ ☑ ☑ ☑ ☑ ☑ ☑
D   □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ☑ ☑ ☑ ☑ ☑ ☑ ☑ ☑
      BBSY  BREQ     BGRIN     IACK      IRQ         AS  DS  LWRD BERR      AM
    BCLR  3      0  3      0 IAI    7              1     1  0 WR DTACK  5           0
CNTL □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ☑ ☑ ☑ □ □ ☑ □ □ □ □ ☑ ☑ ☑
```

Using VM-USB; VM0009 | Idling | Running Firmware Version 8A010903

4.2.2 General Setup

Raw display mode

The states of the VME bus lines are directly mapped onto the LED display, i.e., without being latched or stretched.

Pseudo-DTACK mode

VM-DBA generates a DTACK signal after determining that such has not been issued within a user-defined timeout window.

Halt-enabled mode

The LED display of VM-DBA freezes at the detection of BERR, requiring a reset to re-activate the display.

IRQ-enabled mode

VM-DBA generates an IRQ upon detecting a valid IRQ trigger. Subsequently VM-DBA handles the whole IRQ cycle.

Passive mode

The response of VM-DBA to VME commands is suppressed (with the exception of an unlocking sequence).

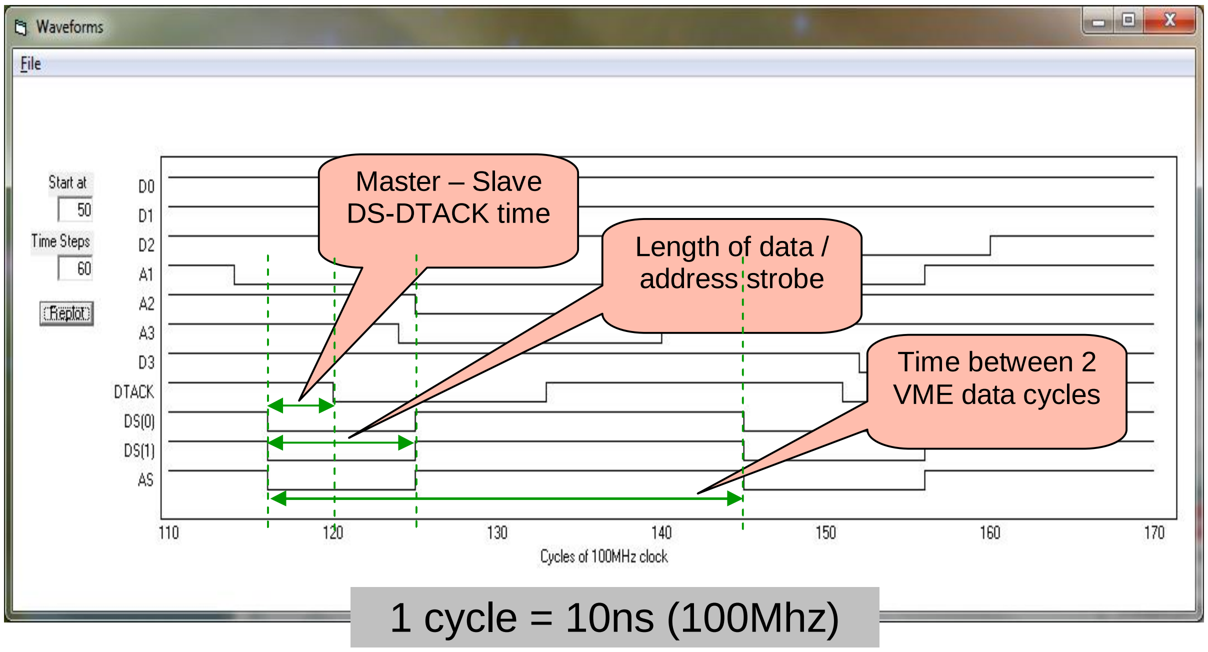
4.2.3 IRQ Setup

The VM-DBA has full interrupt capability in either RORA or ROAC mode. Interrupts can be issued by the NIM/TTL input, by software or internal clock. The IRQ number (1 to 7) and ID can be programmed.

4.2.4 VME Bus Analyzer

The VME bus analyzer allows to sample all VME bus lines as a function of time in order to analyze the sequence and timing of signals which has to match the VME bus specification. The digitization of waveforms is triggered by a programmable selection of conditions as certain addresses or a VME BERR, including an external signal received at the front-panel LEMO connector. Pre-and post trigger sample ranges can be defined with a total length of 2048 samples. The sampling frequency is either 100MHz (10ns steps) or 200MHz (5ns steps).

Example of VME bus lines waveforms:



In case of analyzing rare occurring VME bus failures / states it is possible to configure and start the VM-DBA bus analyzer for a certain condition and leave it without being connected via USB or running the XXWindDBA software. The red **AQ** LED indicates that the VM-DBA is in acquire mode.

When the defined condition is matched the VM-DBA trigger is generated and the data are stored in the internal memory which is shown by a flashing blue LED (**F4**). The data can be then read-out and visualized via VME (VM-USB) or via a direct USB link using the XXWinDBA software.

After reading the data the blue F4 LED will stop flashing and stay on until the acquisition is cleared or a new is started.

Possible Trigger Conditions:

- Hardware input (jumper selectable NIM/TTL)
- Masked IRQ
- Bus Error (BERR)
- Address and Address Modifier combinations:
    - (Valid A) AND NOT (Valid AM)
    - NOT (Valid A) AND (Valid AM)
    - (Valid A) AND (Valid AM)
    - NOT (Valid A) OR NOT (Valid AM)

By selecting a trigger condition the matching input fields will be displayed. All inputs consist of a mask and actual values to match the condition shown in left to the input fields.

Time Base / Range

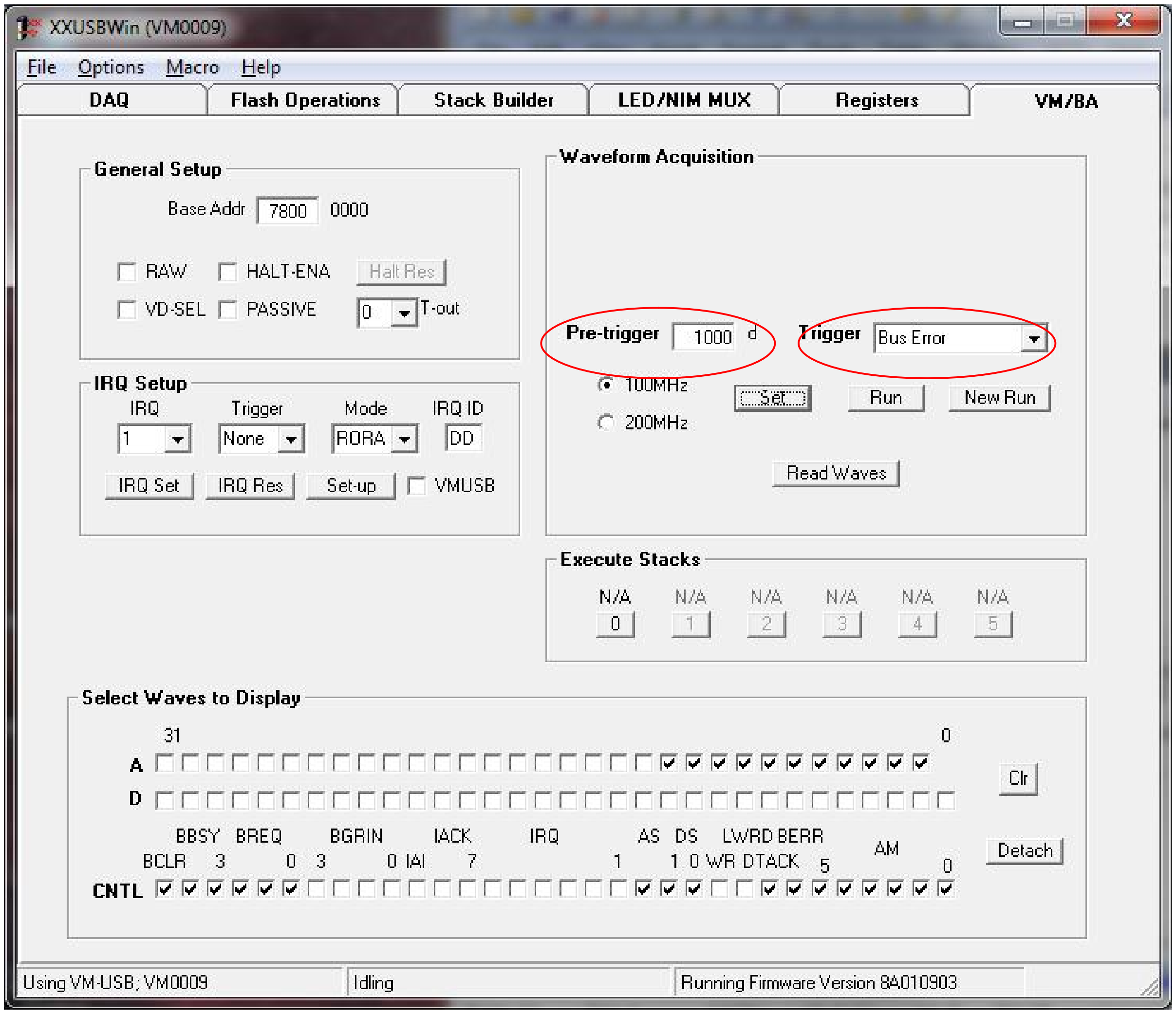
The sampling frequency can be selected to be:

- 100MHz: 10ns steps / 20 µs full range
- 200MHz: 5ns steps / 10 µs full range

The Pre-trigger number of samples can be defined and has to be less than the total number of samples 2048. The remaining samples (2048 – “pre-trigger”) are for the time after the trigger condition. Changing the Pre-trigger setting will require to set and acquire again.

4.2.5 VME Bus Analyzer – Example for BERR condition

1) Set the waveform acquisition trigger condition to “BusError”
2) Define the pre-trigger range to be large enough (example 1000 = 10µs) in order to show the VME cycles which caused the bus error.



Base Addr 7800 0000

RAW ☐ HALT-ENA ☐ VD-SEL ☐ PASSIVE ☐ T-out 0

IRQ 1 Trigger None Mode RORA IRQ ID DD VMUSB ☐

Pre-trigger 1000 d Trigger Bus Error

[✓ 100MHz] 200MHz

Using VM-USB; VM0009 | Idling | Running Firmware Version 8A010903

3) Click on Set to store the conditions

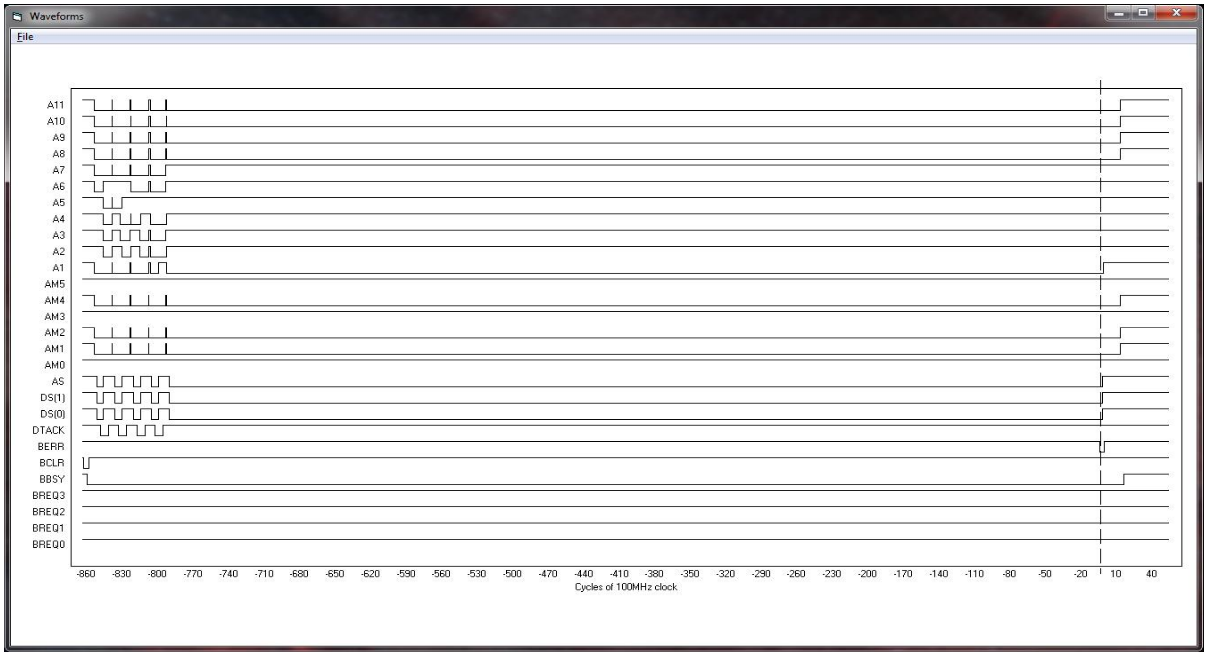
4) Click on Run, the red AQ LED will light up (Run will change to Halt)

5) On a bus error (red BERR will go on) the **AQ** LED will go off and the blue **F4** LED will flash indicating that waveform data for read-out are available

6) Click on Read Waves

7) Select the lines to be shown on the waveform plot. This will open a new window “Waveforms” showing the selected bus line waveforms. With the mouse cursor it is possible to zoom (curser in “cross-mode”) in by selecting a range between time lines. When pointing outside this area (“double-arrow-mode”) the displayed range can be extended by 10 steps (left mouse click) or 100 steps (right mouse click).

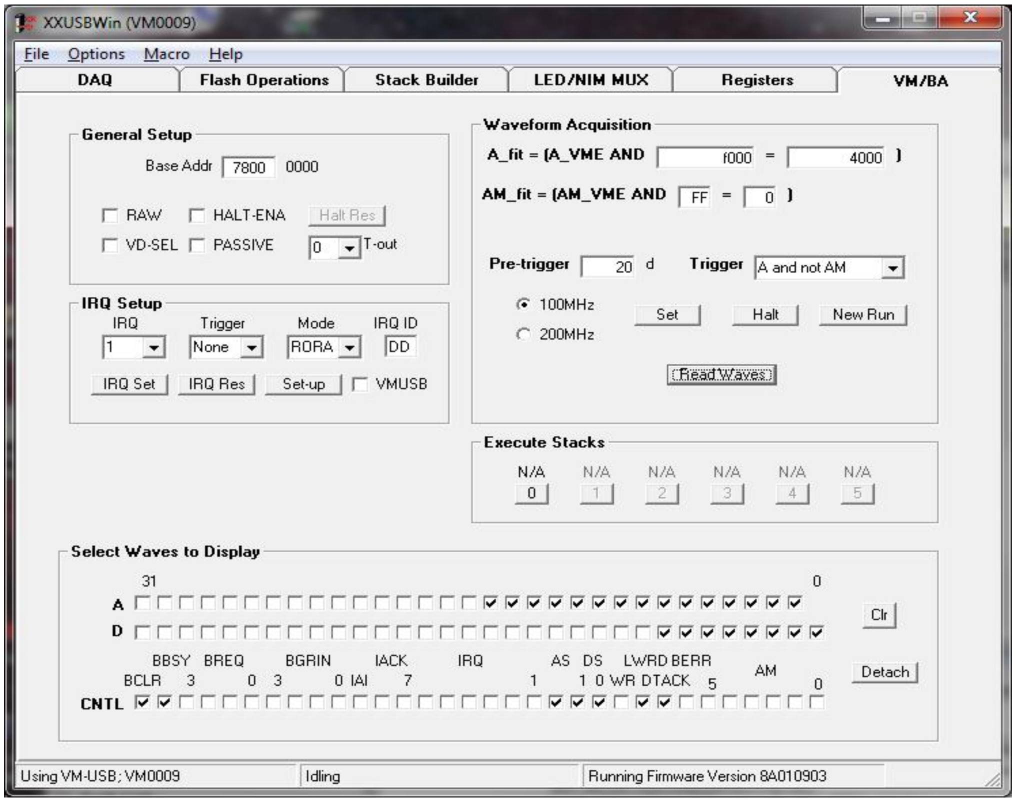
The following picture shows the waveforms for a bus error BERR triggered event. See the BERR at time = 0 (trigger, shown with dashed line) and the last VME cycles about 8µs before the timed out which caused the bus error. The displayed lines can be changed by checking or un-checking the related fields.



4.2.6 VME Bus Analyzer – Example for Address condition

As an example the bus analyzer should be triggered when reading or writing to an address in the range 0x7000 to 0x7ffff, the AM value is ignored:

1) Set the waveform acquisition trigger condition to "A AND NOT AM" and the matching conditions for address (A) and address modifier (AM)



Base Addr 7800 0000

T-out 0

IRQ: 1 Trigger: None Mode: RORA IRQ ID: DD

Waveform Acquisition

A_fit = (A_VME AND f000 = 4000)

AM_fit = (AM_VME AND FF = 0)

Pre-trigger 20 d Trigger A and not AM

(● 100MHz) ○ 200MHz

N/A 0, N/A 1, N/A 2, N/A 3, N/A 4, N/A 5

A 31 … 0: bits 16 to 0 checked (☑), bits 31 to 17 unchecked

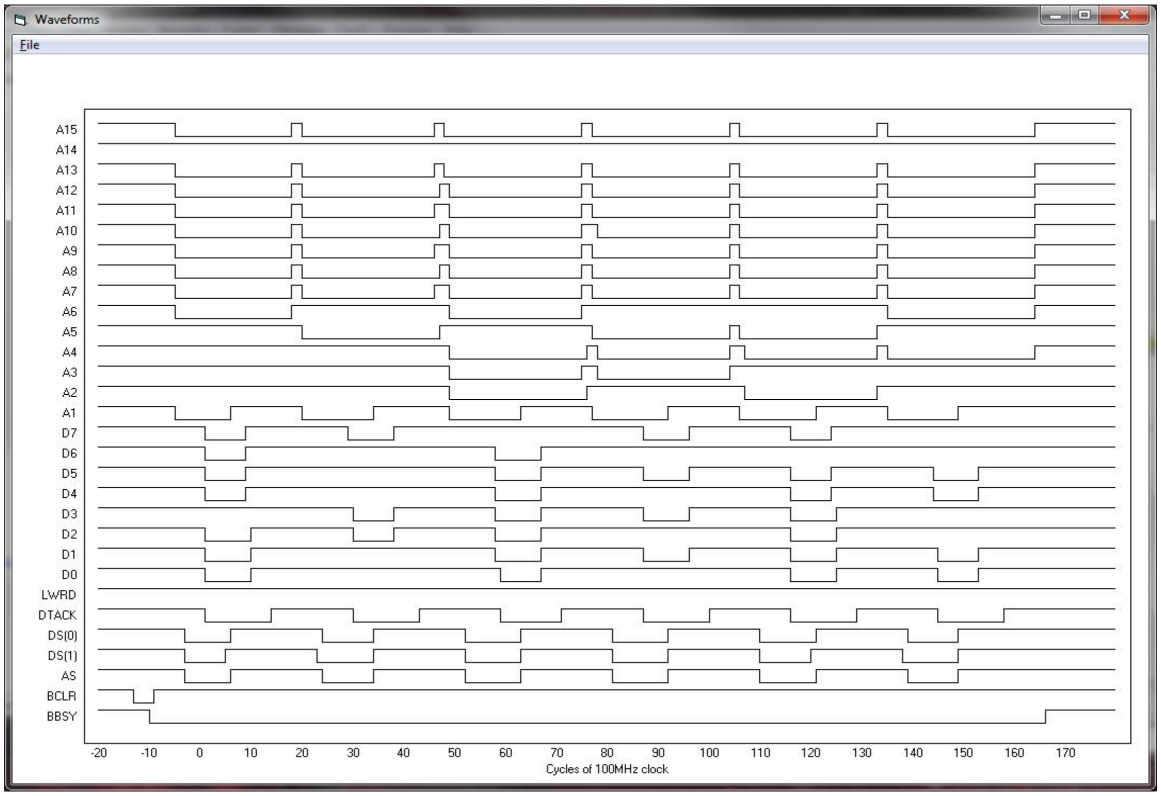
D: bits 7 to 0 checked (☑), bits 31 to 8 unchecked

CNTL: BCLR ☑, BBSY ☑, AS ☑, DS1 ☑, DS0 ☑, WR ☑, DTACK ☑, others unchecked

BBSY BREQ BGRIN IACK IRQ AS DS LWRD BERR AM
BCLR 3 0 3 0 IAI 7 1 1 0 WR DTACK 5 0

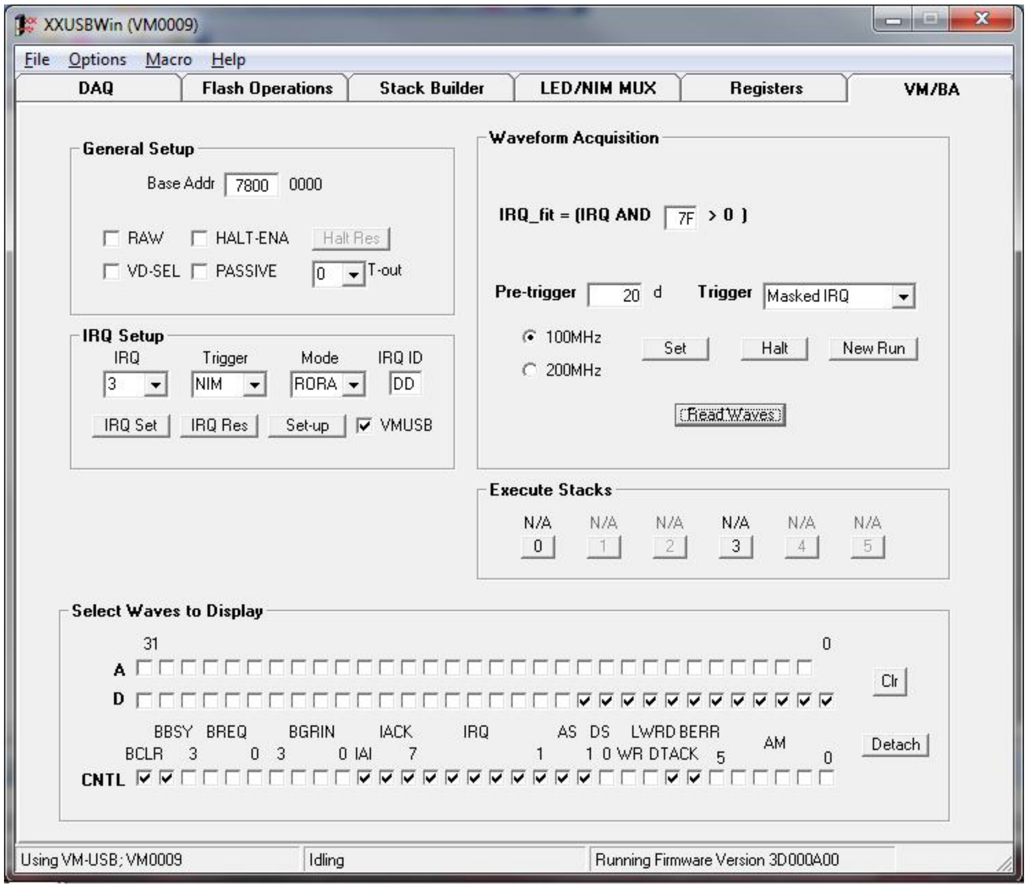
Using VM-USB; VM0009 | Idling | Running Firmware Version 8A010903

2) Define the pre-trigger range (example 20 = 0.2µs) to match the required pre and post trigger time range.
3) Click on Set to store the conditions
4) Click on Run to start the bus analyzer, the red AQ LED will light up (field “Run” will change to “Halt”)
5) In case any VME access will match the defined address range the **AQ** LED will go off and the blue **F4** LED will flash indicating that waveform data for read-out are available
6) Click on “Read Waves”
7) Select the lines to be shown on the waveform plot. This will open a new window “Waveforms” showing the selected bus line waveforms. With the mouse cursor it is possible to zoom (curser in “cross-mode”) in by selecting a range between time lines. When pointing outside this area (“double-arrow-mode”) the displayed range can be extended by 10 steps (left mouse click) or 100 steps (right mouse click).
8) To change the display it is possible to add or remove bus lines by selecting or unselecting those in the “Select Waves to Display” fields.



4.2.7 VME Bus Analyzer – Example for IRQ Interrupt triggered Event

1) Set the waveform acquisition trigger condition to “Masked IRQ”
2) Change the IRQ logic condition in case needed, the default would trigger on any IRQ. (Please note that in this example the IRQ is generated by the VM-DBA on an incoming logic signal into the IN-Input. This is defined in the IRQ setup fields)
3) Click on Set to store the conditions
4) Click on Run, the red AQ LED will light up (Run will change to Halt)



General Setup
Base Addr 7800 0000
☐ RAW ☐ HALT-ENA
☐ VD-SEL ☐ PASSIVE
T-out 0

IRQ Setup
IRQ 3, Trigger NIM, Mode RORA, IRQ ID DD
☑ VMUSB

Waveform Acquisition
IRQ_fit = (IRQ AND 7F > 0)
Pre-trigger 20 d
Trigger Masked IRQ
● 100MHz
○ 200MHz

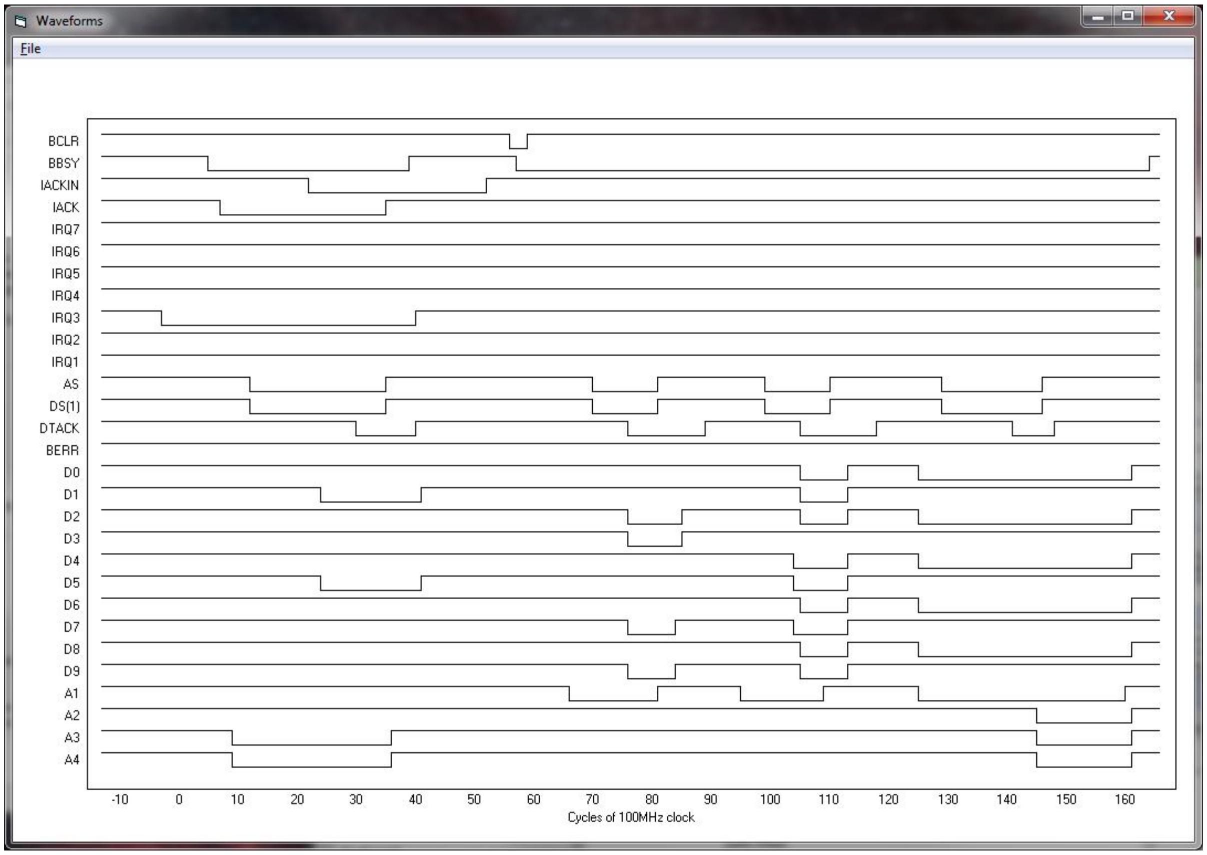
Execute Stacks
N/A 0, N/A 1, N/A 2, N/A 3, N/A 4, N/A 5

Select Waves to Display
A 31 … 0: none selected
D 31 … 0: bits 11 to 0 selected
CNTL: BCLR, BBSY 3 … BBSY 0, BREQ 3 … BREQ 0, BGRIN 3 … BGRIN 0, IAI, IACK, IRQ 7 … IRQ 1, AS, DS 1, DS 0, WR, LWRD, DTACK, BERR, AM 5 … AM 0

Using VM-USB; VM0009 | Idling | Running Firmware Version 3D000A00

5) Any interrupter IRQ 1 to 7 issued on the VME bus will trigger the bus analyzer and as a result **AQ** LED will go off and the blue **F4** LED will flash indicating that waveform data for read-out are available
6) Click on Read Waves
7) Select the lines to be shown on the waveform plot. This will open a new window “Waveforms” showing the selected bus line waveforms. With the mouse cursor it is possible to zoom (curser in “cross-mode”) in by selecting a range between time lines. When pointing outside this area (“double-arrow-mode”) the displayed range can be extended by 10 steps (left mouse click) or 100 steps (right mouse click).

8) In this example IRQ 3 was triggering the analyzer at time “cycles = 0”. Following the IRQ request the interrupter is confirmed by the master (interrupt handler) with IACK. A(0) to A(3) are set by the master with the IRQ value. The interrupt requestor places the vector / ID on the data lines after receiving the IACKIN and asserts a DTACK. Following the BCLR are the VME calls which are issued in response on the interrupt. Please see the detailed description of the VME interrupt handling in the VITA VME bus specification for further details.



5 VME / USB INTERFACE ARCHITECTURE DESCRIPTION

5.1 VME/USB Register Overview

A16 Address Space (BADR16 + Offset)

Offset	WR-Function / DATA	RD-Function / DATA
0	Control Function	Status Register 1
0x10	Interrupt Control + Timeout	
0x20	Software Interrupt Control	
0x30	-	Last Address Register
0x40	-	Last Data Register
0x80	Reset Test Counter	Test Counter

A24 Address Space (BADR24 + Offset)

Offset	WR-Function	RD-Function
0	-	Status Register 2
0x14	Test register	Test register
0x55	Unlock Protected Operation	-

A32 Address Space (BADR32 + Offset)

Offset	WR-Function	RD-Function
0	-	Firmware ID
0x4	Utility and test REG_A	Utility and test REG_A
0x8	Utility and test REG_B	Utility and test REG_B
0xC	Utility and test REG_C	Utility and test REG_C
0x10	Utility and test REG_D	Utility and test REG_D
0x14	Utility and test REG_E	Utility and test REG_E
0x100	-	WVA, VME Address Line Waveform FIFO
0x200	-	WVD, VME Data Line Waveform FIFO
0x300	-	WVC, VME Control Line Waveform FIFO
0x800000	Utility and test FIFO	Utility FIFO
0x1000000	Write-Enable SPI memory	-
0x5000000	Erase SPI memory	-

Note 1: Utility registers can be used for testing of write/read operations, but care should be taken that the stored values don't interfere with their intended use.

5.2 USB Interface

VM-DBA implements an USB interface based on bulk transfers from and to the host computer, via a 512-byte FIFO embedded in the on-board CY7C680013A Cypress USB controller IC. The interface utilizes a protocol similar to that used by VM-USB controller, but supporting only a subset of operations supported by the latter controller. This subset is sufficient to perform efficiently all operations on VM-DBA.

5.3 Communicating with VM-DBA

Communication with the VM-DBA consists of writing and reading packets of data to/from the USB2 port of the VM-DBA using bulk-transfer mode. Borrowing from the USB language, the buffers to be written to the VM-DBA will be called Out Packets, and they are sent to pipe 0 of the USB port. The buffers to be read will be called In Packets, and they are read from pipe 2 of the USB port.

The USB controller IC, when connected to a USB2 port configures packet lengths to 512 bytes in 2-byte words. The Out Packets must be properly formatted to be understood by the VM-DBA firmware and, by the same token, the format of the In Packets retrieved from the VM-DBA must be understood by the user in order to be useful.

In communicating with VM-DBA, it is recommended to use the dynamic link library lixxusb.dll, designed originally for communicating with VM-USB and CC-USB controllers. This library is proven to work under Windows and Linux operating systems, as well as with LabView software. In particular, VM-DBA recognizes two functions of that library designed for interactive executions of properly prepared stacks of VME commands, xxusb_stack_execute and xxusb_longstack_execute. These two functions allow one to

(i) execute 32-bit write and read to/from the registers of VM-DBA in A16, A24, and A32 mode,

(ii) execute both, single and multi-block (up to 1950 blocks at the time) 32-bit A32 writes to the server FIFO for programming the FPGA configuration SPI memories, and

(iii) execute both, single and multi-block (up to 32-blocks of the full content) 32-bit A32 reads from the waveform FIFOs.

It is essential to prepare stacks correctly, as these are interpreted and acted upon by the VM-DBA processor FPGA.

5.3.1 Structure of the Command Stack

A command stack is configured internally as a sequence of 32-bit words beginning with a stack header word and followed by one or more (only in the case of simple write commands) command sequence.

The stack header represents the number of (16-bit) half-words to follow, with every command sequence word representing two half-words, the low 16-bit and the high 16-bit ones.

Every command sequence begins with a Command Header Word with the structure as shown in the table below:

Bits	24 - 31	12 - 23	11	9, 10	8	6, 7	0 - 5
Function	**BLT=64**	*****	**MB**	*****	**NWR**	**DS=00**	**AM**

Where

AM – VME address modifier

0x9 – for simple A32 transfers

0xB – for BLT (block transfer) and multi-BLT A32 transfers

0x39 – for A24 commands, with only simple, non-BLT transfers allowed

0x29 – for A16 commands, with only simple, non-BLT transfers allowed

NWR – 0 for write commands, 1 for read commands

DS – data strobes – both DS0 and DS1 on

MB – 0 for simple and single-BLT transfers, 1 for multi-BLT transfers

BLT – number of 32-bit transfers in a BLT; 64 (0x40) represents 256 bytes.

When multi-BLT transfer is declared, the Command Header Word is to be followed by a NumBlocks word representing number of blocks in the multi-BLT transfer.

Next word in the command sequence is the Address Word representing here the VME address offset only. The base address bits (the 5 most significant bits in the full VME address) are disregarded as it is presumed to apply to the VM-DBA itself.

What is to follow the Address Word depends on the kind of transfer declared in the Header Word, such that

(i) The Address Word is the last word in all READ command sequences – simple, single-BLT, and multi-BLT.

(ii) In a WRITE command sequences, the Address Word is to be followed by the number of Data Words declared in the Command Header Word. This number is 1 for simple WRITE, 64 for single-BLT WRITE, and 64*NumBlocks for multi-BLT WRITE sequence. Note that the latter number may be quite high, equal 64*1950 in multi-BLT WRITE to the utility FIFO.

The following limitations apply to stack preparation:

(i) The command stack may contain many simple WRITE command sequences, but otherwise they are to contain one and only one command sequence.

(ii) BLT and multi BLT WRITE is valid only for the A32 offset of 0x80000 (the utility FIFO)

(iii) BLT and multi-BLT READ is valid only for A32 offsets of 0x100, 0x200, and 0x300 (the waveform FIFOs).

Sample command stacks are illustrated in Appendix X.

5.3.2 Using libxxusb.dll dynamic link library functions

VM-DBA recognizes only USB packets declaring in their header interactive stack execution. These are generated by two library functions, xxusb_stack_execute and xxusb_longstack_execute.

xxusb_stack_execute function

The xxusb_stack_execute function first, sends a buffer to XX-USB, causing the latter to interprete its content as a series of simple and complex VME commands and to actually execute these commands and to write the returned VME data to the USB port FIFO. Then, xxusb_stack_execute reads a buffer from XX-USB, containing the desired VME data.
In the case of VM-DBA passes WRITE data to the registers or utility FIFO or returns the READ data from the registers or the waveform FIFOs to the USB interface.

```
WORD xxusb_stack_execute{
        HANDLE hDevice,
        LPDWORD lpData,
};
```

Parameters
hDevice
[in] Handle to the XX-USB device.

lpData
[in] Pointer to a dual-use variable array. When calling the function, the array contains the data encoding the sequence of desired commands (VME commands for VM-USB and VME commands for VM-USB) to be performed by XX-USB. The first element of the array is the number of bytes. The following command has to be defined similar to the VME / VME command stack (see paragraph 4.5). Upon return, the array contains the VME (VM-USB) or VME (VM-USB) data, respectively.

Return Values
On success, the function returns the number of bytes read from XX-USB. The valid value is twice the number of 16-bit data words returned plus 2 (CC-USB) or 4(VM-USB). The latter “overhead” bytes contain event terminator word (0xFF for VM-USB, and 0xFFFF for VM-USB).
Function returns a negative number on a failure.

xxusb_longstack_execute function

Executes stack array passed to the function and returns the data read from the VME bus
In the case of VM-DBA, passes WRITE data to the utility FIFO of VM-DBA or reads waveform FIFO data and returns these to the USB interface

```
int  xxusb_longstack_execute{

        HANDLE hDevice,
          void *DataBuffer,

          int lDataLen,

          int timeout

};
```

Paramters:

hDevice

[in] Handle to the XX-USB device.

DataBuffer

pointer to the dual use buffer; when calling DataBuffer contains (unsigned short) stack data, with first word serving as a placeholder, upon successful return, DataBuffer contains (unsigned short) VME data

lDataLen

The number of bytes to be fetched from VME bus - not less than the actual number expected, or the function will return -5 code. For stack consisting only of write operations, lDataLen may be set to 1.

Timeout

The time in ms that should be spent tryimg to write data.

Return Values

When Successful, the number of bytes read from xxusb.
Upon failure, a negative number

Remarks

The function must pass a pointer to an array of unsigned integer stack data, in which the first word is left empty to serve as a placeholder.
The function is intended for executing long stacks, up to 4 MBytes long, both "write" and "read" oriented, such as using multi-block transfer operations.

Structure upon call:

DataBuffer(0) = 0(don't care place holder)
DataBuffer(1) = (unsigned short)StackLength bits 0-15
DataBuffer(2) = (unsigned short)StackLength bits 16-20
DataBuffer(3 - StackLength +2) (unsigned short) stack data

StackLength represents the number of words following DataBuffer(1) word, thus the total number of words is StackLength+2

Structure upon return:

DataBuffer(0 - (ReturnValue/2-1)) - (unsigned short)array of returned data when ReturnValue>0

Structure of OUT packets

When the use of the libxxusb dynamic link library is not desired one may pack the stack data into (OUT) packets and bulk-transfer (bulk write) these to the USB port for the subsequent interpretation by the processing FPGA. Since VM-DBA is always expected to return some data in response to an OUT packet, one must subsequently read (via bulk read) an IN packet from the USB interface.

The OUT packet is to be structured in half-words as follows:

1. First half-word is to be always a 0x***C word (16-bit) (* stands for “don’t care”), indicating to the processing FPGA an interactive stack execution type operation, consistent with such for the VM-USB controllers.

2 – N. Command stack half-words

Structure of IN packets

IN packets contain the returned data. For WRITE operations, the returned data = 1. For READ operations, the returned data represent the content of the addressed registers or FIFOs.

5.4 Guide to using the VME bus waveform storage and viewing facility

VM-DBA is capable of storing the states of 95 VME bus lines in 2048 consecutive time steps at a rate of either 100 MS (mega samples per second) or 200 MS. The sampling rate is user-programmable. The waveforms are stored in 3 32-bit wide FIFOs for the subsequent readout either via the VME or the USB interface. The states of individual VME bus lines are represented by bits of these 32-bit words with the following mapping:

(i) **WVA**: VME address bus lines A(1 to 31) are mapped in a natural order to bits of the waveform address word WFA(1 to 31). When the waveform acquisition is active, the latter words are stored in the Address FIFO at the base address of BADR32 + **0x100**.

(ii) **WVD**: VME data bus lines D(0 to 31) are mapped in a natural order onto bits of the waveform data word WFD(0 to 31). When the waveform acquisition is active, the latter words are stored in the Data FIFO at the base address of **BADR32 + 0x200**.

(iii) **WVC**: Other 32 VME bus lines are mapped onto bits of the waveform control word WFC in a way illustrated below. When the waveform acquisition is active, the latter words are stored in the Control FIFO at the base address of **BADR32 + 0x300**.

WFC	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
VME	**DTACK**	**BERR**	**AM5**	**AM4**	**AM3**	**AM2**	**AM1**	**AM0**

WFC	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
WME	**IRQ3**	**IRQ2**	**IRQ1**	**AS**	**DS1**	**DS0**	**NWR**	**LWRD**

WFC	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
VME	**BG1**	**BG0**	**IACKIN**	**IACK**	**IRQ7**	**IRQ6**	**IRQ5**	**IRQ4**

WFC	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**
VME	**BCLR**	**BBSY**	**BR3**	**BR2**	**BR1**	**BR0**	**BG3**	**BG2**

The waveform acquisition proceeds as follows:

Upon the setup of the acquisition parameters and entering the acquisition regime, VM-DBA writes the waveform words WFA, WFD, and WFC into their respective FIFOs configured for a circular continuous operation. Thus, at any moment of time, these FIFOs store waveforms in 2048 preceding time steps. Upon detection of the trigger signal parts of the FIFOs are drained so as to preserve the desired number of pre-trigger steps, while the freed-up space is filled with post-trigger waveforms, until the FIFOs are filled. Obviously, one cannot store more than 2048 pre-trigger steps. When so desired, one may forgo not only all pre-trigger steps, but also an additional 2048 steps, i.e., effectively commencing acquisition 2048 time steps after the detection of the trigger signal.

Subsequently, one may read out the contents of the three waveform FIFOs via a sequence of 32 32-bit block read commands, terminated by the BERR signal indicating that the particular FIFO has been emptied.

5.5 Setup of the waveform acquisition

The setup of the waveform acquisition entails (i) selection of the sampling frequency of either 100 MHz or 200 MHz, (ii) defining of the trigger signal for the commencement of the waveform storage cycle, (iii) defining of the start time step with respect to the trigger signal, and (iv) activation of the continuous circular storage of the waveforms in the three waveform FIFOs, in anticipation of the trigger signal.

The waveform setup parameters are stored in sections of the utility registers REGA, REGB, REGC and REGD structured as shown in tables below.

REG_A at the base address of BADR32 + 4

Bits	**31 - 1**	**0**
Function	**TRA**	**WFFR**

TRA – Trigger Reference Address
WFFR – Waveform FIFOs Reset when set. Must be reset for FIFOs to be active.

REG_B at the base address of BADR32 + 8

Bits	**5, 4**	**0**
Function	**MSPS**	**RAWDIS**

MSPS – Sampling frequency code
  00 – 100 Msps
  01 – 200 Msps
RAWDIS – when 1, front panel LEDs display live state of the VME bus lines (no latching).

REG_C at the base address of BADR32 + 12 (0xC)

Bits	**31 - 1**
Function	**AMASK**

AMASK – Active trigger address bit**s.**

REG_D at the base address of BADR32 + 16 (0x10)

Bits	**13 - 8**	**6 - 0**
Function	**TRAM**	**AMMASK/IRQMASK**

TRAM – Trigger Reference Address Modifier
AMMASK – Active trigger address modifier bits when AM-based trigger.
IRQMASK – Trigger IRQ Mask when IRQ-based trigger

REG_E at the base address of BADR32 + 20 (0x14)

Bits	**18 - 16**	**11**	**10 - 0**
Function	**TRSEL**	**PREPOST**	**START**

TRSEL – Trigger Select Code
0 – None
1 – IN (NIM/TTL)
2 – Masked IRQ
3 – Bus Error (BERR)
4 – (Valid A) AND NOT (Valid AM)
5 – NOT (Valid A) AND (Valid AM)
6 – (Valid A) AND (Valid AM)
7 – NOT (Valid A) OR NOT (Valid AM)
PREPOST – when set, storage of the waveforms commences STRT cycles after the detection of the trigger signal. When reset, storage commences STRT cycles before the detection of the trigger signal

NOTE: Continuous waveform acquisition into (circularly configured) FIFO's is activated by setting TRSEL > 0. It is then terminated at the detection of a valid trigger signal.

Here are the steps to be made to start a waveform acquisition process:

1. Select the desired sampling frequency by writing either 0 (100 MS) or 16 (200 MS) into REG_B at the base address 8.
2. When the intended trigger code is 01 or 03, proceed to point xx.
3. When the intended trigger code is 2, do the following
    a. Write the trigger reference address TRA into REG_A at the base address of 4
    b. Write the trigger IRQ mask TIRQMASK into REG_D at the bas address of 16 (0x10).
    d. Proceed to point 5.

    Boolean equations for the valid trigger ValTrigg is here

    ValTrig = ((IRQ$_{\mathrm{VME}}$ AND IRQMASK) >0)

4. When the intended trigger code is greater than 3, do the following
    a. Write the trigger reference address TRA into REG_A at the base address of 4

b. Write the address mask AMASK into REG_C at the bas address of 12 (0xC).

c. Write the trigger reference address modifier TRAM and the address modifier mask AMMASK into REG_D at the base address of 16 (0x10)

d. Proceed to point 5.

Boolean equations for the valid triggers ValTrigg are here

ValTrig = AS AND (ValA AND (NOT ValAM)) for the trigger code 4

ValTrig = AS AND ((NOT ValA) AND ValAM) for the trigger code 5

ValTrig = AS AND (ValA AND ValAM) for the trigger code 6

ValTrig = AS AND ((NOT ValA) OR (NOT ValAM)) for the trigger code 7

where

ValA = (TRA = (A_{VME} AND AMASK)) ,

ValAM = (TRAM = (AM_{VME} AND AMMASK), and

AS is the address strobe

5. Write the start time of the waveform storage relative to the trigger signal and the trigger selection code into REG_E at the base address 20 (0x14).

   At this moment the red LED labeled AQ, indicating continuous waveform acquisition should light up.

5.6 Processing of the acquired waveforms

When in continuous waveform acquisition mode and upon the detection of a valid trigger signal, VM-DBA completes storage of waveforms as specified in the setup. At this time the red AQ LED should turn off and the blue LED labeled F4 should begin blinking. If this is not the case, begin the cycle from the beginning. Otherwise proceed with the readout of the three waveform FIFOs. It is advisable to utilize the block read or multi-block read as foreseen in the VM-USB protocol. When the FIFOs are emptied, the F4 LED should become steady blue. Subsequently, use a software of your choice to visualize the waveforms.

6 SPECIAL MODES OF OPERATION

For test purposes, VM-DBA may be set to operate in modes other than the default mode with the display reflecting the states of the VME bus lines latched or stretched at the last VME activity, and the unit responsive when addressed via VME or USB interface. The modes are largely similar to those offered by the predecessor module VDIS-2, except that they are here controlled by internal registers rather than by front-panel switches.

There are five special modes of operation of VM-DBA, controlled by registers in the A16 address space:

1. Raw display mode, where the states of the VME bus lines are directly mapped onto the LED display, i.e., without being latched or stretched.
2. Pseudo-DTACK mode, where VM-DBA generates a DTACK signal after determining that such has not been issued within a user-defined timeout window.
3. Halt-enabled mode, when the LED display of VM-DBA freezes at the detection of BERR, requiring a reset to re-activate the display.
4. IRQ-enabled mode, where VM-DBA generates an IRQ upon detecting a valid IRQ trigger. Subsequently VM-DBA handles the whole IRQ cycle.
5. Passive mode, where response of VM-DBA to VME commands is suppressed, with the exception of an unlocking sequence

The modes are reasonably non-exclusive with the exception of the passive mode, when VM-DBA is nonresponsive to conventional commands.

The modes are controlled by 3 D32-mode registers in the A16 address space, the Control Register 1, Interrupt control register, and interrupt software trigger register. The structure of these registers is illustrated in tables below.

Control Register 1 at the base address of 0 (A16 mode)

Bits	**6**	**5**	**2**	**0**
Function	PASSIVE	RAW	HENA	PDTACK
LED label	PASS	CONT	H	VD-E

Interrupt Control Register at the base address of 16 (0x10) (A16 mode)

Bits	**15,14**	**13,12**	**10 - 8**	**7 - 0**
Function	IRQTRSEL	TOUT	IRQ-ID	IRQVECT
LED label	I-EN	-	-	-

IRQTRSEL – IRQ trigger selection code:

0 – IRQ facility disabled
1 – IN (NIM/TTL)

2 – Soft (by writing to interrupt software trigger register)
3 – 1kHz internal clock.

TOUT – Timeout code for pseudo-DTACK generator.
0 – 4 µs
1 – 3 µs
2 – 2 µs
3 – 1 µs

IRQ-ID – IRQ level (1 – 7)

IRQVECT – 8-bit identifier

Interrupt Software Trigger Register at the base address of 32 (0x20) (A16 mode)

Bits	**1**	**0**
Function	IRQ-RES	IRQ_SET
LED label	-	F1

Both bits are write-only toggle bitts.

7 PROTECTED OPERATIONS

There are four kinds of operations that can be performed only after a special “unlock” sequence is issued. These are:

- (i) Erase SPI memory
- (ii) Program SPI memory
- (iii) Reboot
- (iv) Exit from passive mode.

For the first three of the above operations, the requirement for an “unlock” serves the purpose of avoiding inadvertent corruption of the configuration SPI memory or inadvertent rebooting of VM-DBA.

As far as the exit from the passive mode is concerned, unlocking is necessitated by the fact that while in passive mode, VM-DBA is not responsive to regular commands but still checks for the receipt of the “unlock” sequence. Then, upon detecting “unlock” it responds to the subsequent “exit from passive” command and only to that one.

7.1 The Unlock Sequence

The ***unlock sequence*** consists of two consecutive A24/D32 “write” commands to the address offset of 0x55554 of data words, first 0x55555555 and then, 0xAAAAAAAA. It is valid only for the operation that directly follows the sequence.

8 SPI MEMORY OPERATIONS

There are two SPI memories on-board to store two FPGA configuration files. The FPGA first boots from the memory selected by a respective jumper. Upon a successful boot, VM-DBA allows the following operations involving the SPI memory:

- (i) Selecting a particular target SPI memory, regardless of the default-boot jumper setting.
- (ii) Write enabling a particular SPI memory.
- (iii) Erasing the selected memory (a protected operation requiring prior unlocking).
- (iv) Checking the status of the erase operation.
- (v) Programming the selected memory – memory must be erased prior to programming (both operations require prior unlocking)
- (vi) Checking the status of programming.
- (vii) Rebooting from the selected SPI memory (a protected operation requiring prior unlocking).

8.1 Selecting the Target SPI Memory

The memory is selected by writing its code bits (0x20000000 – for protected, 0x30000000 for main) into utility register Reg_b at the (32-bit) base address of 8. The selection should be made right before the intended SPI memory operation, as Reg_b may be used also for other unrelated operation.

8.2 Write Enable SPI Memory

To erase or program an SPI Memory, one must first write-enable it. For the programming operation, this is performed by the FPGA programming routine automatically prior to programming every individual 256-byte long block of memory. For the bulk-erase operation, one must issue explicitly the write-enable command by writing (anything) to the register at A32 address offset of 0x1000000.

8.3 Erasing the Selected SPI Memory

To erase a SPI memory one needs to do the following

(i) Select the target SPI memory; if this is the protected one, set the protector jumper to the “unprotect” position.
(ii) Issue the “write enable” command,
(iii) Execute the “unlock” sequence,
(iv) Immediately following the “unlock” sequence, write 0xAAAAAAAA to the A24 address offset of 0x55554 (same as that of the “unlock” sequence).

The last of the above steps triggers the bulk-erase cycle that is wholly controlled by the SPI memory IC itself. One may then check on the status of the operation by reading the register at A32 address offset of 0x1000000 and checking the value of bit 0. If the latter value is 1, the cycle is still in progress, otherwise it has completed.

8.4 Programming the SPI Memory

The programming proper of a section of the SPI memory is performed automatically by a dedicated FPGA routine, after certain conditions are met. It is performed in units of 64 256-byte long blocks using 65 consecutive D32/A32 BLT “write” operations to the (128k-deep, 32-bit wide) utility FIFO at A32 offset of 0x800000. Each block of data must be preceded by its starting address in the SPI memory – a 32-bit word, an overhead that for 64 blocks amounts to a full extra (65-th) block. The XXDBAWin software employs here the “proprietary” multiblock “write” operation with 30x65=1950 BLT transfers. The programming sequence involves the following steps:

(i) Write to utility register Reg_a at offset = 8 the number of BLT “writes”, NumBLT, to the utility FIFO that triggers the automatic transfer of the content of that FIFO into the SPI memory. Note that NumBLT must be a multiple of 65 for the reasons (overhead) discussed earlier.

(ii) Issue the “unlock” sequence.

(iii) Prepare NumBLTx64 32-bit unsigned data array to be transferred to the utility FIFO.

(iv) Perform NumBLT BLT “writes” of the prepared data array into the utility FIFO.

The last step triggers automatic transfer of the section of the configuration data previously written into the FIFO into the selected SPI memory, starting at memory locations specified within the prepared data array.

The structure of the D32 data array to be written into the utility FIFO is as follows:

StartAddress1	- start address of the following data in the SPI address space
DataWord1	unsigned 32-bit data word composed of the consecutive 4 bytes of the XILINX configuration *.bit file with little-endian mapping

.

.

DataWord64

StartAddress2 = StartAddress1 + 256 (the SPI address is in bytes)

DataWords

.

.

StartAddressLast = StartAddress1 + 256*NumBLT*64/65

DataWords

The XILINX byte-wide configuration *.bit file must be stripped of its header section up to the first 0xFF.

Obviously, one must repeat the above operations (i)-(iv) until the full configuration data file has been transferred. However, before proceeding with the consecutive operation, one must make sure that the previous one has terminated. One does this by verifying the status bit 13 (0x2000) in the A24 Status Register 2 at offset 0. This bit, when set, indicates operation in progress.

The programming of the SPI memory is a complex operation requiring expertise. One is advised to use the XXDBAWin software to perform this operation either directly via the USB port of VM-DBA or via a VM-USB controller.

9 IRQ SERVICE TEST

VM-DBA allows one to test the integrity of the IRQ handling net in the VME crate by issuing, upon detecting a valid trigger signal, an IRQ on any of the seven IRQ1 – IRQ7 lines. Subsequently it responds to the IAQ/IAQIN/AS inquiry by the IRQ handler by placing on VME bus data lines 0 – 7 a preprogrammed IRQ vector accompanied shortly after by DTACK. This IRQ vector is assumed to be recognizable to the IRQ handler, which after terminating the IRQ handshaking cycle performs the duty assigned to this particular IRQ ID and this particular IRQ vector.

VM-DBA implements two modes of IRQ resetting, RORA (Reset on Register Access) and ROAC (Reset on Access). The RORA mode requires the VME controller to reset IRQ by writing 2 into Software Interrupt Register at A16 base address of 32 (0x20). When configured for ROAC mode, VM-DBA resets IRQ upon detecting valid IRQ vector on VME Bus address lines 1-3, accompanied by AS, IACK, and IACKIN.

Setting up the IRQ testing involves writing into the IRQ register at the A16 address offset of 16 (0x10) the IRQ setup word structured as shown in table below

Structure of the IRQ setup word

Bits	**14, 15**	**12, 13**	**11**	**8 - 10**	**0 - 7**
Function	**TrigSel**	**TOUT**	**ROAC**	**IRQID**	**IRQVECT**

IRQVECT - 8-bit IRQ vector identifying the requestor VM-DBA to the IRQ handler
IRQID – 3-bit code identifying the IRQ bus line 1 – 7
ROAC – when 1, ROAC mode is selected. Otherwise RORA
TOUT – unrelated to the IRQ facility time-out code described in Section 6.
TrigSel – IRQ trigger selection code

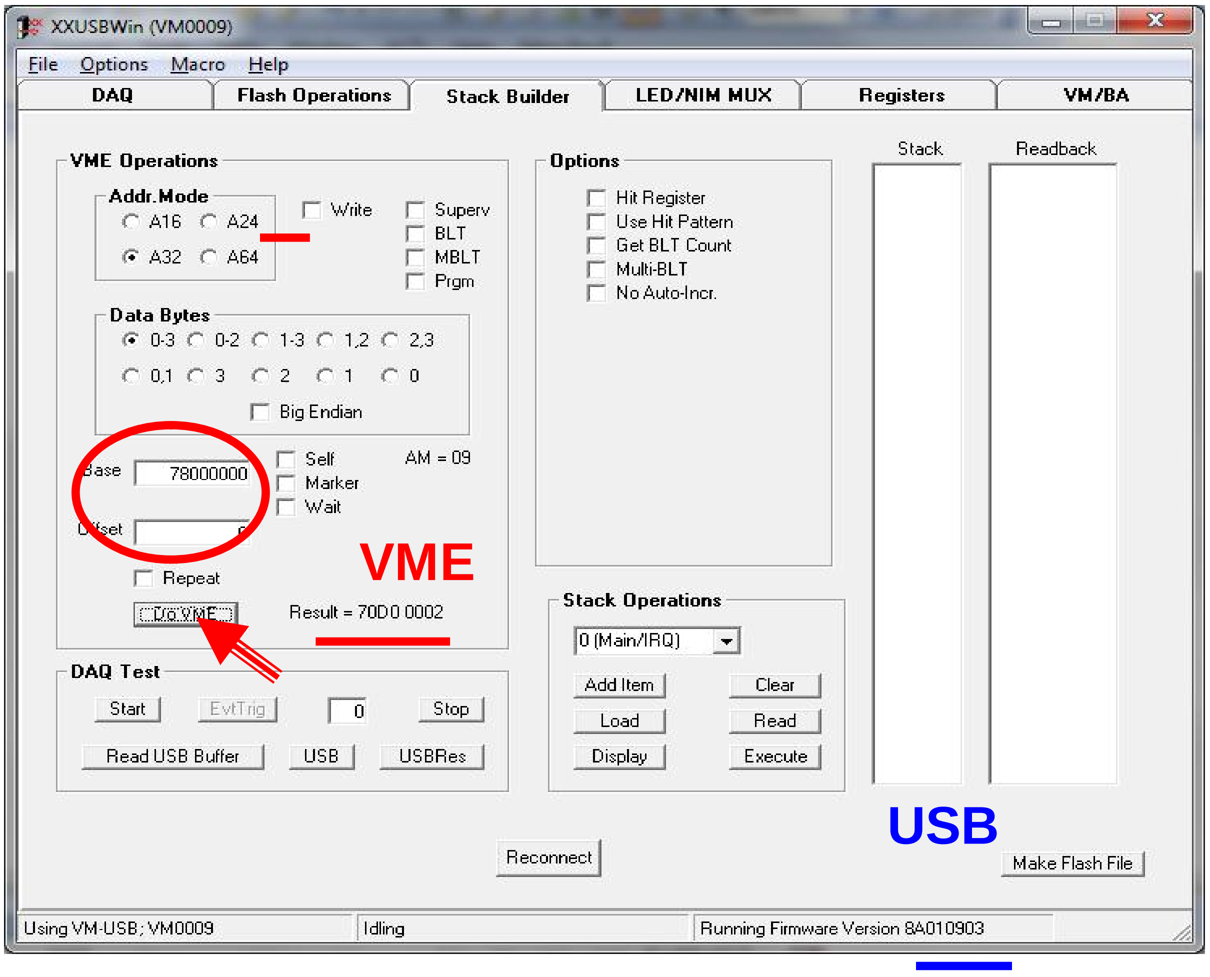
0 – IRQ generation disabled
1 – NIM/TTL
2 – Software
3 – Periodic at 1 kHz.

10 FIRMWARE UPGRADE PROCEDURE

The VM-DBA firmware can be updated through either the USB port or via VME by using the VM-USB controller. Please use the XXDBAWin program and get the latest firmware file from the www.wiener-d.com web site.

1. Check Base address of VM-DBA (following example shown for factory default Base address 0x78000000).
Note! The Base address is only needed when updating the firmware via VME!

2. Check current Firmware Version (VME only!, see marks in red):
Power VME crate,
Run XXDBAWin
Read A32 from Address 0x7800 0000 (hex) should return firmware ID



Addr.Mode: A32
Data Bytes: 0-3
Base: 78000000
AM = 09
Result = 70D0 0002
Stack Operations: 0 (Main/IRQ)

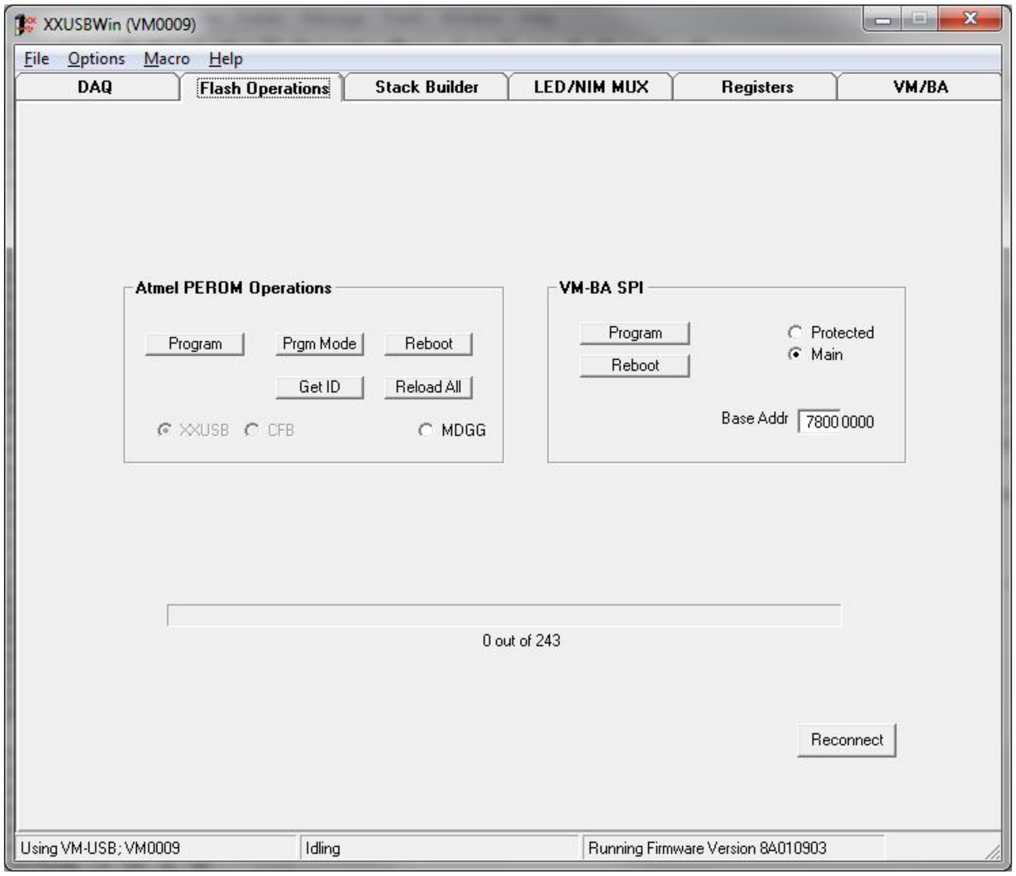
Using VM-USB; VM0009 | Idling | Running Firmware Version 8A010903

➔ Response: 70D0 0002
Note: In case of a direct USB connection to the VM-DBA the firmware is shown at the bottom status line (shown in blue).

3. Program new Firmware

Go to “Flash Operations” tab and see right side (VM-BA SPI)

Select “Main” SPI memory location



VM-BA SPI: Protected [✓ Main]

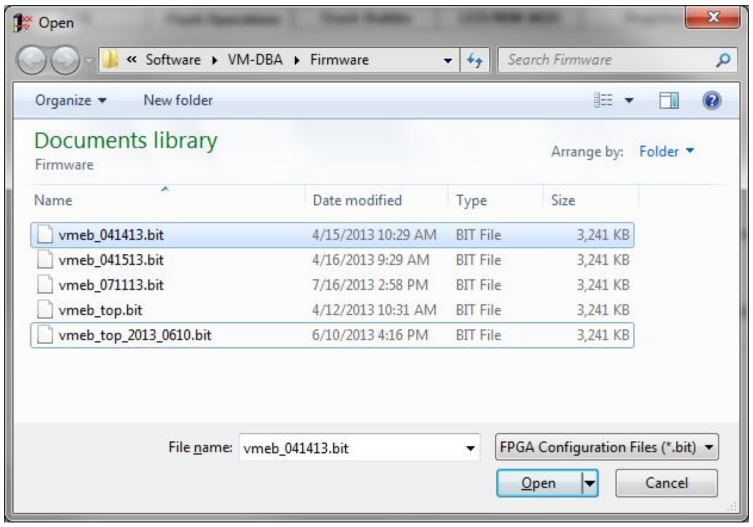
Base Addr 7800 0000

0 out of 243

Using VM-USB; VM0009 | Idling | Running Firmware Version 8A010903

For upgrade via VME type in value for A32 Base Address (**7800 0000 for factory default**). This value is not used and can be ignored for upgrade via USB.

Click on program and select the right firmware file

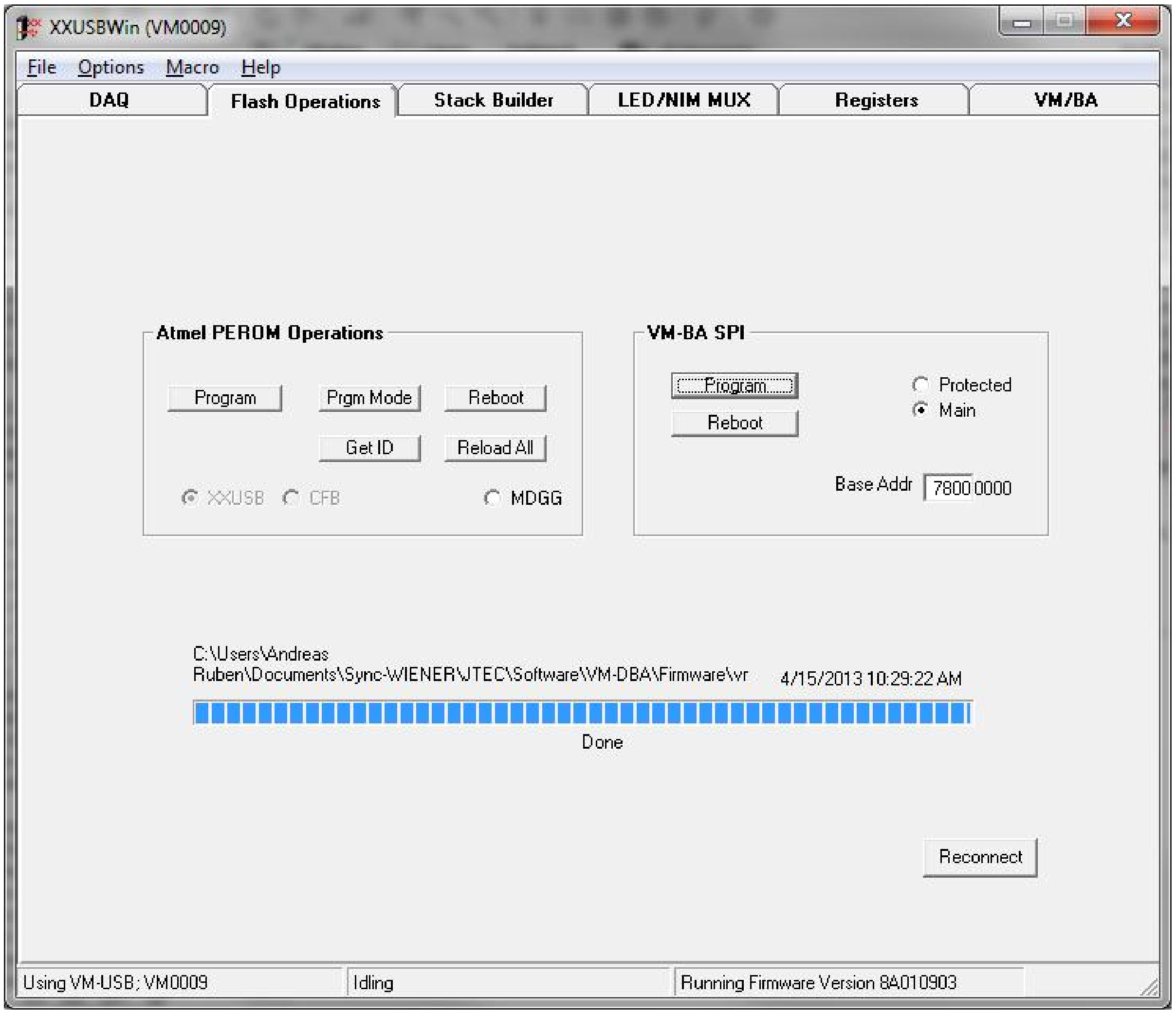


Name	Date modified	Type	Size
vmeb_041413.bit	4/15/2013 10:29 AM	BIT File	3,241 KB
vmeb_041513.bit	4/16/2013 9:29 AM	BIT File	3,241 KB
vmeb_071113.bit	7/16/2013 2:58 PM	BIT File	3,241 KB
vmeb_top.bit	4/12/2013 10:31 AM	BIT File	3,241 KB
vmeb_top_2013_0610.bit	6/10/2013 4:16 PM	BIT File	3,241 KB

File name: vmeb_041413.bit

FPGA Configuration Files (*.bit)

When erasing the old firmware and programming flashing LED's on the VM-DBA will indicate the upgrade.



Base Addr 78000000

C:\Users\Andreas Ruben\Documents\Sync-WIENER\JTEC\Software\VM-DBA\Firmware\vr 4/15/2013 10:29:22 AM

Done

Using VM-USB; VM0009 | Idling | Running Firmware Version 8A010903

4. Reboot VM-DBA:

Reboot the VM-DBA by clicking on Reboot or power cycle the VME crate.

11 APPENDIX A: SAMPLE VM-DBA COMMAND STACKS

Stack entries below are shown in hexadecimal half-words to be packed into array arguments of xxusb_stack_execute (array of longwards) or xxusb_longstack_execute (array ou unsigned integers).

A. Simple A32 WRITE command
Writing 0x12345678 into utility register Reg_a
The following stack is target for xxusb_stack_execute and must be therefore packed into an array of 32-bit integers – on half-word per one 32-bit integer.

1.	0007	7 half-words to follow
2.	0000	
3.	0009	AM=0x9 and non-BLT A32 WRITE
4.	0000	
5.	0004	address offset for Reg_a
6.	0000	
7.	5678	data – low half-word
8.	1234	data – high half word

Note: xxusb_stack_execute returns 1 on success.

B. Stack of two simple A32 Write commands
Writing 0x2222111 into Reg_a and 0x44443333 into Reg_b
The following stack is target for xxusb_stack_execute and must be therefore packed into an array of 32-bit integers – on half-word per one 32-bit integer.

1.	000D	13 half-words to follow
2.	0000	
3.	0009	AM=0x9, non-BLT A32 write
4.	0000	
5.	0004	address offset for Reg_a
6.	0000	
7.	1111	data – low half-word
8.	2222	data – high half word
9.	0009	AM=9, non-BLT A32 WRITE
10.	0000	
11.	0008	address offset for Reg_b
12.	0000	
13.	3333	data – low half-word
14.	4444	data – high half-word

Note: xxusb_stack_execute returns 1 on success.

C. Multi-BLT WRITE

Writing 1950 blocks into the utility FIFO – note that a multi-BLT WRITE of this length can be handled only by xxusb_longstack_execute function and not xxusb_stack-execute. The following stack is to be packed in an array of unsigned 16-bit integers.

0.	0000	placeholder half-word for xxusb_longstack_execute
1.	CF07	
2.	0003	0x3CF07 = 249607 half-words to follow
3.	080B	multi-BLT A32 WRITE
4.	4000	0x40 = 64 words per BLT
5.	079E	0x79E = 1950 blocks to be written
6.	0000	
7.	0000	low half-word of the address offset for the utility FIFO
8.	0800	high half-word of the address offset for the utility FIFO
9 – 249609.		half-words of the data to be written

Note: xxusb_longstack_execute returns 1 on success.

D. Multi-BLT READ from the Address waveform FIFO

Reading 32 blocks from the FIFO at address offset of 0x100 targeting xxusb_longstack_execute. Stack to be packed into an array of unsigned 16-bit integers.

0.	0000	placeholder
1.	0007	7 half-words to follow
2.	0000	
3.	0D0B	multi-BLT A32 READ
4.	4000	0x40 = 64 words per BLT
5.	0020	0x20 = 32 blocks to be read
6.	0000	
7.	0100	address offset for the Address waveform FIFO
8.	0000	

Note: xxusb_longstack_execute returns 2048 32-bit data words.