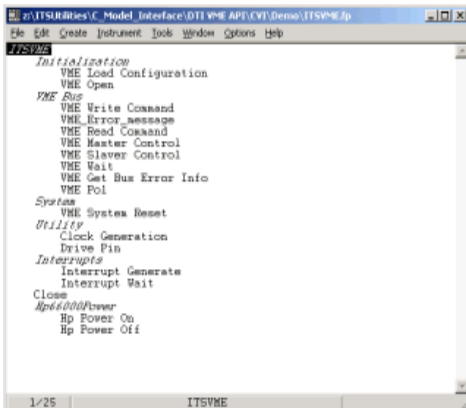


Digital Test Instrument VME C-Model Interface

The VME API provides test developers with a high level-programming environment suited for developing or re-hosting test programs. Advanced test sets become more feasible due to the API's ability to be used in both C and C++ based development environments.



VMEbus API C-Model Interface integrated into LabWindows CVI, as a test instrument - with easy to use function calls.

The VMEbus API allows your Digital Test Instrument (DTI) to communicate with VME circuit card assemblies (CCA's) using standard VME protocol. The API emulates VME activity using internal state machines.



Integrated Test Solutions
585 Shaker Road East Longmeadow, MA 01028
Phone: (413) 525-2888 Fax: (413) 525-2051
www.itsus.com

VMEBus

VMEbus C-Model Interface

C-Model Interface

VMEbus Application Program Interface for Digital Test Instruments



Integrated Test Solutions

The complex VME bus communication has been implemented as an easy to use application program interface (API) using a Digital Test Instrument (DTI) as the VMEbus sub system. The API allows the DTI to communicate with VME circuit card assemblies (CCA's) using standard VME protocol. The API emulates the VME bus activity via an internal state machine and manifests the bus activity dynamically through the digital test instrument. The API contains functions related to the use of the VMEbus master and provides a high level means of communicating with any VME based device. The VME API provides test developers with a high level programming environment to develop or re-host tests.

Table of Contents

| | |
|---|-------------------------------------|
| Abstract..... | Error! Bookmark not defined. |
| 1.0 Design | 3 |
| Names | 3 |
| Return codes..... | 3 |
| Limitations | 3 |
| 1.1 Implementation | 3 |
| 2.0 Application Program Interface..... | 4 |
| Overview..... | 4 |
| DTI VME Functionality Supported:..... | 4 |
| 2.1 VME General Function Tree and Prototypes..... | 5 |
| 2.1.1 Function Tree Layout:..... | 5 |
| 2.1.2 VME_Error_message..... | 6 |
| 2.1.3 ClockGen | 6 |
| 2.1.4 VMEClose..... | 7 |
| 2.1.5 VMEDrivePin | 7 |
| 2.1.6 VMEGetBusErrorInfo..... | 8 |
| 2.1.7 VMEInterruptGenerate | 8 |
| 2.1.8 VMEInterruptWait..... | 8 |
| 2.1.9 VMELoadConfig | 9 |
| 2.2.0 VMEMaster..... | 10 |
| 2.2.1 VMEOpen | 11 |
| 2.2.2 VMESPol | 13 |
| 2.2.3 VMERead | 14 |
| 2.2.4 VMEReset..... | 15 |
| 2.2.6 VMEWait..... | 16 |
| 2.2.7 VMEWrite..... | 16 |
| 3.0 Utility Program | 17 |
| 3.1 VMEbus Configuration Utility | 18 |
| 4.0 Typical Program Flow | 19 |

1.0 Design

The DTI VME application program interface has been designed using the C/C++/C# programming environment. This type of interface allows the use of many different libraries that can be directly linked into the test program for a complete testing integration environment. Special user defined pins are accessible and debugging of the bus is made easier by the interface's ability to report tester pin states on a channel and pattern level. The following are design and implementation notes.

Names

- The API uses readable and meaningful names for all of its functions, as well as the common prefix "VME".

Return codes

- All functions of the API return an unambiguous return code.
- The return code is equal to 0 if the function has terminated without error.
- The return code is different from 0 in case the function terminated with an error.

Limitations

- No Read-Modify-Write functions are defined in the API. Those can be added later if needed.
- Bus arbitration functions have been defined but can be added later as needed.
- Address Only Cycles are defined but not implemented in the API
- Interrupt Service routines are defined but not implemented in the API
- Master Cycles only are supported – slave mode can be integrated later as needed.
- Clock Generators are defined and can later be implemented as needed.

1.1 Implementation

The following are related to the implementation of the API:

1. Layered implementation

The implementation of the API can use low-level libraries and/or system-level drivers if necessary. The number of different libraries or drivers and the dependencies on other external libraries shall, however, be minimized.

2. Utility programs

The implementation of the API should be accompanied by a utility program which is used to configure the VMEbus for pin maps, initial drive states, channel loading etc.

3. Bus error handling

The implementation of the API shall encapsulate the VMEbus error handling. At the level of single-word read and write access, the user shall have the choice to check the bus error status or to ignore it.

5. Logging

The implementation of the API shall log serious errors with a central logging facility, e.g. a global file or kernel messages. The implementation of the API shall also log events of the VMEbus of general interest with the logging facility.

7. Language binding

C# .NET environment was chosen for the language binding of the API

The environment has also been implemented using Lab Windows CVI as an intermediate until the .NET framework system has been fully implemented in the industry.

2.0 Application Program Interface

Overview

The following are an overview of all types and functions defined in the VMEbus API:

DTI VME Functionality Supported:

The VME Bus API will support the following bus activities:

- DTI VME Open
- DTI VME Close
- DTI VME Write
- DTI VME Read
- DTI VME Reset
- DTI VME Block Transfer
- DTI VME Interrupt
 - DTI VME Interrupt Wait
 - DTI VME Interrupt Generate
- Bus Arbitration
 - REQUESTER
 - ARBITER
- Periodic Clocks
- Failure Detection

NOTE:

- If not stated otherwise, all functions of this API are non-blocking, i.e. they return immediately indicating an error code if necessary. Wherever functions are blocking, i.e. waiting on external events, e.g. end of block transfer ,or VMEbus interrupt, this is stated explicitly.
- The return values of all function of this API can be used for comparison, after the error code has been converted to an error number. The return value VME_SUCCESS can always be used for comparison.

2.1 VME General Function Tree and Prototypes

ITSVME

The functions appear in alphabetical order, with a description of the function and its C syntax, a description of each parameter, and a list of possible error codes.

2.1.1 Function Tree Layout:

| Class/Panel Name: | Function Name: |
|-----------------------|----------------------|
| Initialization | |
| ▪ Load Configuration | VMELoadConfig |
| ▪ Open VME | VMEOpen |
| VME Bus | |
| ▪ VME Write Command | VMEWrite |
| ▪ VME_Error_message | VME_Error_message |
| ▪ VME Read Command | VMERead |
| ▪ ARB Control Master | VMEMaster |
| ▪ ARB Control Slave | VMESlave |
| ▪ Wait | VMEWait |
| ▪ Get Bus Error Info | VMEGetBusErrorInfo |
| ▪ VME Pol | VMEPol |
| System | |
| • System Reset | VMEReset |
| Utility | |
| • Clock Generation | VMEClockGen |
| • Drive Pin | VMEDrivePin |
| Interrupts | |
| ▪ Interrupt Generate | VMEInterruptGenerate |
| ▪ Interrupt Wait | VMEInterruptWait |
| Close | |
| ▪ VMEClose | VMEClose |

The following functions are in alphabetical order.

2.1.2 VME_Error_message

```
void VME_Error_message (char *errorMessage);
```

Purpose

The VME_Error_message routine will return the most recent function call or VME error

Return Value

Completion Codes

VI_SUCCESS

The function completed successfully.

Parameter List

errorMessage

Variable Type char (passed by reference)

2.1.3 ClockGen

```
ViStatus VMEClockGen (int testerChannel, float frequency);
```

Purpose

The VMEClockGen() function can be used to program a DTI channel to function as an oscillator at program frequencies.

Return Value

Completion Codes

VI_SUCCESS

The function completed successfully.

Parameter List

testerChannel

Variable Type int

frequency

Variable Type float

2.1.4 VMEClose

ViStatus VMEClose (void);

Purpose

The VMEClose() releases all resources which were allocated in a VMEOpen() function call and closes the VMEbus library/driver. This function is the last function of the API to be called by the application program.

Return Value

Completion Codes

VI_SUCCESS

The function completed successfully.

Error Codes

2.1.5 VMEDrivePin

ViStatus VMEDrivePin (int testerChannel, int driveDuration,
 int driveState);

Purpose

The VMEDrivePin () will allow the use to drive any defined channel to any particular state.

Return Value

Completion Codes

VI_SUCCESS

The function completed successfully.

Parameter List

testerChannel

Variable Type int

driveDuration

Variable Type int

driveState

Variable Type int

2.16 VMEGetBusErrorInfo

ViStatus VMEGetBusErrorInfo (void);

Purpose

The VMEGetBusErrorInfo() function returns information on the last cycle completed determine if the BUSERR flag was raised.

2.1.7 VMEInterruptGenerate

ViStatus VMEInterruptGenerate (int level);

Purpose

The VMEInterruptGenerate() function generates a VMEbus interrupt at Specified level with desired vector. This function can be used in order to send an interrupt to another VMEbus interrupt handler.

Parameter List

level

Variable Type int

2.1.8 VMEInterruptWait

ViStatus VMEInterruptWait (int level, int timeOut);

Purpose

The VMEInterruptWait() function waits until an interrupt associated to interrupt is received or until the time-out has elapsed, whichever occurs first. Time_out is an estimate for the time-out period in milliseconds. The value 0 is used to bypass the time-out mechanism and to return immediately, indicating the status of the interrupt. The value -1 is used to bypass the time-out mechanism and to wait until an interrupt is received. After a VMEbus interrupt has been received interrupt_info contains the information on the VMEbus interrupt actually received. Depending on time_out and on the return code, interrupt_info might be empty.

Parameter List

level

Variable Type int

Interrupt Level 0 - 6

timeOut

Variable Type int

The value 0 is used to bypass the time-out mechanism and to return immediately, indicating the status of the interrupt.

The value -1 is used to bypass the time-out mechanism and to wait until an interrupt is received.

2.1.9 VMELoadConfig

ViStatus VMELoadConfig (char *configFilename);

Purpose

This function loads the configuration file into memory, and applies all configuration options to the tester.

Note* This function is called as part of the VMEOpen Funtion and is not intended to be called as a stand alone function.

The configuration file is in the standard INI file format.

Sections Are:

[Levels] - This section specifies the two level sets used by the tester.

[Timing Set] - This section specifies that values used for each timing set.

[AddressBusX] - Where X is the bus line

[DataBusX] - Where X is the bus line

[Control] - VME Control Signals

[Misc] - Non-VME Pins to be defined here

Example

[Addressbus2]

Name=VMEP1A29

Channel=245

Capture=Window

Connect=Closed

Hybrid=NA

Impedance=50Ohm

Mode=Dynamic

Format=NRet

Level=0

Phase=0

Window=0

Load=On

Parameter List

configFilename

Variable Type char *

Use full path to configuration file name.

2.2.0 VMEMaster

ViStatus VMEMaster (void);

Purpose

Normally the DTI is always the master and this command is only necessary if switching roles.

2.2.1 VMEOpen

```
ViStatus VMEOpen (char *configFilename, ViRsrc resourceName,  
                 ViBoolean IDQuery, ViBoolean resetInstrument);
```

Purpose

The VMEOpen() function opens the VMEbus library/driver and allocates the resources required to use the VMEbus. It also loads and configures all setups defined in the configuration utility. This function must be called prior to any other function of the VMEbus API.

Parameter List

configFilename

Variable Type char *

resourceName

Variable Type ViRsrc

Identifies the M9-Series DTI with which to establish communications, using one of the following formats:

TERM9::logical_address Identifies the M9-Series DTI whose Central Resource Board (CRB) has the specified logical address.

TERM9::chassis_id, slot_id Identifies the M9-Series DTI whose CRB is in the specified chassis and slot.

TERM9::#sn Identifies the nth software M9-Series DTI in the current configuration. The first software M9-Series DTI is #s0.

TERM9::#n Identifies the nth hardware M9-Series DTI in the current configuration; the first hardware M9-Series DTI is #0. If TERM9::#n cannot be found, TERM9::#sn is used instead.

TERM9:: or TERM9 or TERM9::* or TERM9::undefined_string
Identifies the first M9-Series DTI in the current
configuration; is equivalent to TERM9::#0.

IDQuery

Variable Type ViBoolean

In-system verification mode, one of:

VI_TRUE (1)

Performs in-system verification when session communications are established.

VI_FALSE (0)

Bypasses in-system verification when session communications are established.

resetInstrument

Variable Type ViBoolean

DTI reset mode, one of:

VI_TRUE (1)

Initializes and resets the M9-Series DTI when session communications are established.

VI_FALSE (0)

Does not initialize and reset the M9-Series DTI when session communications are established.

Return Value

Completion Codes

VI_SUCCESS

The function completed successfully.

VI_WARN_NSUP_ID_QUERY

Identification query is not supported.

TERM9_WARN_SOFTWARE_SIM

Init failed to find hardware. Hardware is being simulated in software.

Error Codes

VI_ERROR_FAIL_ID_QUERY

The instrument identification query failed.

TERM9_ERROR_INIT

DTI initialization failed.

2.2.2 VMEPol

ViStatus VMEPol (int reqLevel, int dataSize, long address_Hex, int addressModifiers, int expectData, int maxCycles, int mask_Hex);

Purpose

The VMEPol command will read an address maxcycles times waiting for the expected data to arrive. The parameter *mask* can be used to only look at particular bits in the data field specified..

Status will return 1 if expected data was not present
return 0 is expected data was present.

Parameter List

reqLevel

Variable Type int

dataSize

Variable Type int

address_Hex

Variable Type long

addressModifiers

Variable Type int

expectData

Variable Type int

Block Size is used when a selected Address Modifiers describes a block mode transfere.

Pass 1 if no block transfere

maxCycles

Variable Type int

Block Size is used when a selected Address Modifiers describes a block mode transfere.

Pass 1 if no block transfere

mask_Hex

Variable Type int

2.2.3 VMERead

ViStatus VMERead (int reqLevel, int dataSize, long address_Hex, int addressModifiers, int blockSize, long data_Hex[]);

Purpose

This function will issue a VME Read from the board attached to the tester. All arguments must be filled in. Data will be read from the address specified.

Status will be returned with a zero for a success and an error code on a failure.

Parameter List

reqLevel

Variable Type int

dataSize

Variable Type int

address_Hex

Variable Type long

addressModifiers

Variable Type int

blockSize

Variable Type int

Block Size is used when a selected Address Modifiers describes a block mode transfere.

Pass 1 if no block transfere

data_Hex

Variable Type long []

2.2.4 VMEReset

ViStatus VMEReset (int systemResetPatterns);

Purpose

This function will hold system reset low for the specified number of patterns - Then release

Parameter List

systemResetPatterns

Variable Type int

2.2.5 VMESlave

ViStatus VMESlave (int slaveControlCommand);

Parameter List

slaveControlCommand

Variable Type int

2.2.6 VMEWait

ViStatus VMEWait (int patterns_toWait);

Purpose

This function will insert number of patterns as a wait.
The implementation of the wait will depend on the size.

If the wait is < 1000 patterns then a simple CREPEAT pattern modifier
will be inserted.

If the wait is larger then 1000 patterns then a CLOOP CREPEAT and ENLOOP
pattern modifiers will be inserted.

Parameter List

patterns_toWait

Variable Type int

2.2.7 VMEWrite

ViStatus VMEWrite (int reqLevel, int dataSize, long address_Hex,
 int addressModifiers, int blockSize,
 long data_Hex[]);

Purpose

This function will issue a VME Write to the board attached to the tester. All arguments must be filled in. The data specified will be written to the address specified.

Status will be returned with a zero for a success and an error code on a failure.

Parameter List

reqLevel

Variable Type int

dataSize

Variable Type int

address_Hex

Variable Type long

addressModifiers

Variable Type int

blockSize

Variable Type int

Block Size is used when a selected Address Modifiers describes a block mode transfere.

Pass 1 if no block transfere

data_Hex

Variable Type long []

3.0 Utility Program

The DTI VMEbus utility program accompanies the API implementation: a utility program to configure the VMEbus initial settings must accompany the VMEbus API. The program has been written in C# and will utilize the XML file format. The CVI Lab Windows implementation will use the INI format. The INI format is limited in how the

data can be structured and is only used as an interim process until the .NET system is fully employed. Version 1.0 employed the INI format without the utility program.

3.1 VMEbus Configuration Utility

The VMEbus configuration utility (“*DTI_VME_Config*”), is used to configure all parameters necessary to use the DTI_VMEbus API.

The internal structure is patterned as follows:

VME_BUS Structure

- PIN_PARAMETERS
- LEVELS
- TIMING

LEVELS – Indexed by Level Set 0 or 1

- VIH
- VIL
- VOH
- VOL
- IOH
- IOL
- VCOM
- SLEW

TIMING – Indexed by Timing Set (0 – 255)

- Clock Period
- Phase (0 - 3) Assert Return Values
- Window (0 - 3) Open Close Values

PIN_PARAMETERS – Indexed by VME Pin (i.e. VMEP1A1)

- Pin Name
- Pin Alias
- DTI Channel Used
- Relay Connect/Disconnect
- Hybrid Open/Closed
- Level Set
- Impedance (Zout)
- Phase
- Window
- Format
- Load On/Off

All of the above values will be stored in the VME Bus configuration file generated by the API Bus Definition Utility. When the *config_DTI_vme* function is called, these values will be loaded into memory and applied, where applicable, to the tester. These values will remain constant until overridden.

The above bus structure will have an internal private function (*private_init_VMEbus*) to initialize their contents to defaults. The config routine will then load the config file and override the values if present. All functions not accessible to the end user will begin with *private*. They are to be used only internally by API functions.

The API Utility itself will allow the addition, deletion, and modification of all values defined above via a GUI. This utility will in essence allow the user to defin

4.0 Typical Program Flow

A typical VME API program will consist of the following steps:

1. Initialize the VME API
2. Load the VME Configuration file
3. Set any user defined options on the DTI
4. Issue VME Bus Commands (VME READ WRITE RESET)
5. When Finished Close the Handles obtained from the Initialization of instruments