ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Tektronix VX5530* at our website: ***Click HERE***



VX5530/VX5535
VXIbus System Controller
C Programmers Manual

070-7438-00 (Book 3 of 3)
First Printing JAN 1990

Tektronix®
COMMITTED TO EXCELLENCE

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OLRMGetNumAttr

OLRMGetNumAttr

Purpose: Returns requested numerical information about a specific device.

Syntax:

```
unsigned short far pascal OLRMGetNumAttr(devname, ula,
                                         attr);
char far *devname;        /* device name */
unsigned short ula;       /* unique logical address */
unsigned short attr;      /* attribute */
```

Description: The device can be addressed by symbolic name or logical address.

Attribute	Source
OLRM_CLASS	VXIbus ID register
OLRM_ADDRESS_MODE	"
OLRM_MANUFACTURER	"
OLRM_REQ_MEMORY	VXIbus device-type register
OLRM_MODEL	"
OLRM_ADDRESS_BASE	VXIbus offset register
OLRM_MEMORY_TYPE	VXIbus memory attribute register
OLRM_SPEED	"
OLRM_LOG_ADDR	devtab
OLRM_SLOT	"
OLRM_CMDR	"
OLRM_BID	VXIbus ID register
OLRM_BDT	VXIbus device-type register
OLRM_BSC	VXIbus status register
OLRM_BSO	VXIbus offset register
OLRM_BAT	VXIbus memory attribute register
OLRM_BPR	VXIbus message protocol register
OLRM_BRE	VXIbus message response register
OLRM_BMH	VXIbus message data-high register
OLRM_BML	VXIbus message data-low register

If the device is a VXIbus device, acquisition of most of the attributes causes a VXIbus access.

NOTE

The available attributes cover both fields as well as whole registers. The encoding is the same as defined in the VXIbus specification (e.g., OLRM_CLASS returns a value in the range 0..3).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The OLRM_LOG_ADDR attribute denotes the logical address of the device. The OLRM_SLOT attribute denotes the slot in which the device resides. The OLRM_CMDR attribute denotes the logical address of the device's commander. Every device has a commander. The commander of the top level commander is itself. The OLRM_BID, OLRM_BDT, OLRM_BSC, OLRM_BSO, OLRM_BAT, OLRM_BPR, OLRM_BRE, OLRM_ BMH, and OLRM_BML attributes denote the value of the entire VXIbus register.

Return Value: As defined in the VXIbus specification.

Errors: The value returned is 0xFFFF.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OLRMGetSlotName

OLRMGetSlotName

Purpose: Finds the name of the device in a particular slot.

Syntax:

```
short far pascal OLRMGetSlotName(slot, devname);
unsigned short slot;     /* slot number */
char far *devname;       /* device name in the slot */
```

Description: The caller is responsible for allocating at least 13 bytes for the devname parameter.

Return Value: Returns `VC_SUCCESS` if successful.

Errors: Returns `ERR_FAIL` on error.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OLRMGetStringAttr OLRMGetStringAttr

Purpose: Returns requested ASCII information about a specific device.

Syntax:

```
char far * far pascal OLRMGetStringAttr(devname, ula,
                                        attr, string);
char far *devname;        /* device name */
unsigned short ula;       /* unique logical address */
unsigned short attr;      /* attribute */
char far *string;         /* device information */
```

Description: The device can be addressed by symbolic name or logical address.

Attribute	Source
OLRM_DEVICE_NAME	devtab
OLRM_MANUFACTURER	devtab
OLRM_MODEL	devtab

These attributes are the symbolic values as reported by the SURM. The caller is responsible for allocating at least 13 bytes for the fourth parameter (the output string). The value is placed in this string and the address of this string is the returned value.

Return Value: The address of the output string.

Errors: If an error occurs, a NULL pointer is returned.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OLRMRename

OLRMRename

Purpose: Changes the symbolic name of a device.

Syntax:

```
char far * far pascal OLRMRename(devname, ula, newname);
char far *devname;          /* device name */
unsigned short ula;         /* unique logical address */
char far *newname;          /* new name of device */
```

Description: The device can be addressed by its current symbolic name or logical address. If the device is found, its name is changed to that of `newname` (or the first 12 characters of `newname` if larger than 12) and the returned value is identical to the newname parameter. If the device cannot be found, the returned value is NULL.

If the device being renamed is a VXIbus message-based device and MDS is installed, this function also calls the Message Delivery System's `MDSDelete Device` and `MDSCreateDevice` functions to notify MDS of the new name.

NOTE

`OLRMRename` *does not change the hardware configuration file* **config.hwf**. *The name change is lost when the machine is shut down or rebooted.*

Return Value: The return value is identical to the `newname` parameter.

Errors: If an error occurs, a NULL pointer is returned.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Section 6
TekVICS Advanced Trigger Functions

Introduction

The Advanced Trigger Functions (ATF) provide advanced trigger, signal, and timing features within a VXIbus mainframe, as well as the necessary real time links between multiple VXIbus subsystems.

The user should be familiar with the functionality of the Advanced Trigger Functions. See Appendix A of the VX5530/VX5535 Users Manual for details.

Examples showing how to use the TekVICS Advanced Trigger Functions are provided at the end of this section.

NOTE

All of the ITB lines must be driven by a signal. None of the ITB lines can be left in an indeterminate state (neither high nor low), because resonance may occur in the other routed signals. One way to solve this is to find an unused STAR line and route it to all the unused ITB lines, then disable the receiver of the STAR line and enable the drivers of all the unused ITB lines.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATF Functions

Following is a functional listing of TekVICS Advanced Trigger Functions. Detailed descriptions of all functions follow later in this section.

ATF Functional Listing

Input/Output Unit	Configuration
ATFEnOutBNC( ) ATFEnInBNC( ) ATFOutBNCRt( ) ATFOutBNCBStat( )	ATFChkOption( ) ATFCkInstall( )
Delay Unit	**Logical Unit**
ATFSetDelayU( ) ATFEnDelayU( )	ATFSetLogicIn( ) ATFSetLogicOut( ) ATFEnLogicIn( ) ATFEnLogicOut( )
Clock Unit	**STARbus routing unit**
ATFSelSysClk( )	ATFEnSRRec( ) ATFEnSRDrv( ) ATFSTARRoute( ) ATFSTARuproc( )
Power Temperature Monitor	**Protocol Unit**
ATFPowerTempStat( )	ATFSetProtoU( ) ATFEnProtoU( ) ATFSelTTLTRG( ) ATFSelECLTRG( ) ATFSendTrg( )

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Alphabetical Function Listing

Following is an alphabetical listing and a brief purpose statement for each function. Functions are listed in detail in the remainder of this section.

ATFCkInstall	Checks if ATF Device Driver is installed.
ATFChkOption	Checks current hardware configuration.
ATFEnDelayU	Enables/disables Delay Units.
ATFEnSRDrv	Enables/disables STARbus routing unit drivers.
ATFEnSRRec	Enables/disables STARbus routing unit receivers.
ATFEnInBNC	Enables/disables Input BNCs.
ATFEnLogicIn	Enables/disables Logical Unit input lines.
ATFEnLogicOut	Enables/disables Logical Unit output.
ATFEnOutBNC	Enables/disables Output BNCs.
ATFEnProtoU	Enables/disables Protocol Units.
ATFOutBNCRt	Routes signal from an ITB line to Output BNC.
ATFOutBNCBStat	Reads the current Output BNC B status.
ATFPowerTempStat	Measures power supply voltage and the board temperature inside the VX5530/VX5535.
ATFSelECLTRG	Selects ECL Trigger Line.
ATFSelSysClk	Selects internal/external clock.
ATFSelTTLTRG	Selects TTL Trigger Line.
ATFSendTrg	Sends internal μP trigger.
ATFSetDelayU	Sets up Delay Unit.
ATFSetLogicIn	Sets up Logical Unit input.
ATFSetLogicOut	Sets up Logical Unit output.
ATFSetProtoU	Sets up Protocol Unit.
ATFSTARRoute	Routes a signal through a STARbus routing unit.
ATFSTARuproc	Sets STARbus routing unit external trigger.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFCkInstall

ATFCkInstall

Purpose: Checks whether the Advanced Trigger Functions Device Driver is installed.

Syntax:

```
short far pascal ATFCkInstall( );
```

Description: This function checks whether the Advanced Trigger Functions Device Driver (**atf.sys**) is installed.

Return Value: Returns `VC_SUCCESS` if the driver is installed and `ERR_FAIL` if the driver is not installed.

Errors: None.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFChkOption

ATFChkOption

Purpose: Checks the current hardware configuration.

Syntax:

```
short far pascal ATFChkOption( );
```

Description: Checks the current VX5530/VX5535 hardware configuration and saves it until the system is rebooted. This function can be used to check the number of STARbus routing units supported (1 or 3).

When the **atf.sys** device driver is installed, the driver parameters specify a default hardware configuration, and that default configuration is loaded internally on reboot. After **ATFChkOption** is executed, the current hardware configuration is saved internally. However, the default configuration will be restored each time the VX5530/VX5535 is rebooted.

NOTE

While this function is executing, it will output a signal to the Output BNC B.

Return Value: Returns one of the following constants which are defined in the **atf.h** include file.

HWNOTAV	Hardware not available
ONESRCONFIG	Only one STARbus routing unit is supported for STAR lines on VXIbus slots 2-5 (Option 01).
FULLCONFIG	All three STARbus routing units are supported.

Errors: None.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFEnDelayU

ATFEnDelayU

Purpose: Enables or disables signal routing through the Delay Unit.

Syntax:

```
short far pascal ATFEnDelayU(delayunit, mode);
unsigned short delayunit;      /* delay unit */
unsigned short mode;           /* enable or disable */
```

Description: Enables or disables signal routing through the Delay Unit.

The following constants are defined in the **atf.h** include file.

Legal values for `delayunit` are:

DELAY 1	Delay Unit 1
DELAY 2	Delay Unit 2

Legal values for `mode` are:

ON	Enable Delay Unit
OFF	Disable Delay Unit

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

ERR_PARA	Parameter error
ERR_HW	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFEnSRDrv

ATFEnSRDrv

Purpose: Enables or disables STARbus routing unit drivers.

Syntax:

```
short far pascal ATFEnSRDrv(unit, mode);
unsigned short unit;     /* STARbus routing unit */
unsigned short mode;     /* channels to be enabled */
```

Description: Enables or disables the drivers of a STARbus routing unit. Enabling a driver causes the routed signal to be output on the corresponding channel.

`unit` specifies which STARbus routing unit is to be used. See legal values below. STARbus routing units 1 and 2 are not supported for Option 01 hardware.

`mode` specifies the channels to be enabled (up to 16 on each STARbus routing unit). By ORing the constants, multiple channels can be enabled at the same time.

The following constants are defined in the **atf.h** include file.

Legal values for `unit` are:

SR0	STARbus routing unit 0
SR1	STARbus routing unit 1
SR2	STARbus routing unit 2

Legal values for `mode` are:

ITB0-7	ITB lines 0..7, available for all three Routing Units
STARX2-5	STARX 2..5 (STARbus routing unit 0)
STARY2-5	STARY 2..5 (STARbus routing unit 0)
STARX6-9	STARX 6..9 (STARbus routing unit 1)
STARY6-9	STARY 6..9 (STARbus routing unit 1)
STARX1,STARX10-12	STARX 1, STARX 10..12 (STARbus routing unit 2)
STARY1,STARY10-12	STARY 1, STARY 10..12 (STARbus routing unit 2)
0	Disables all drivers of the STARbus routing unit selected

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

ERR_PARA	Parameter error
ERR_HW	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFEnSRRec

ATFEnSRRec

Purpose: Enables or disables STARbus routing unit receivers.

Syntax:

```
short far pascal ATFEnSRRec(unit, mode);
unsigned short unit;    /* STARbus routing unit */
unsigned short mode;    /* channels to be enabled */
```

Description: Enables or disables the receivers of a STARbus routing unit. Enabling a receiver causes the input signal to be routed through the corresponding unit.

`unit` specifies which STARbus routing unit is to be used. See legal values below. STARbus routing units 1 and 2 are not supported for Option 01 hardware.

`mode` specifies the channels to be enabled (up to 16 on each STARbus routing unit). By ORing the constants, multiple channels can be enabled at the same time.

The following constants are defined in the **atf.h** include file.

Legal values for `unit` are:

SR0	STARbus routing unit 0
SR1	STARbus routing unit 1
SR2	STARbus routing unit 2

Legal values for `mode` are:

ITB0-7	ITB lines 0..7, available for all three Routing Units
STARX2-5	STARX 2..5 (STARbus routing unit 0)
STARY2-5	STARY 2..5 (STARbus routing unit 0)
STARX6-9	STARX 6..9 (STARbus routing unit 1)
STARY6-9	STARY 6..9 (STARbus routing unit 1)
STARX1,STARX10-12	STARX 1, STARX 10..12 (STARbus routing unit 2)
STARY1,STARY10-12	STARY 1, STARY 10..12 (STARbus routing unit 2)
0	Disables all receivers of the STARbus routing unit selected

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

ERR_PARA	Parameter error
ERR_HW	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFEnInBNC

ATFEnInBNC

Purpose: Enables or disables Input BNCs.

Syntax:

```
short far pascal ATFEnInBNC(bnc, mode);
unsigned short bnc;      /* Input BNC */
unsigned short mode;     /* enable or disable */
```

Description: Enables or disables Input BNCs. Enabling an input BNC causes the external signal to be connected to the appropriate Instrument Trigger Bus (ITB) line.

The following constants are defined in the **atf.h** include file.

Legal values for `bnc` are:

`IBNCA`	Input BNC A
`IBNCB`	Input BNC B

Legal values for `mode` are:

`ON`	Enable Input BNC
`OFF`	Disable Input BNC

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

`ERR_PARA`	Parameter error
`ERR_HW`	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFEnLogicIn

ATFEnLogicIn

Purpose: Enables or disables Logic Unit input line.

Syntax:

```
short far pascal ATFEnLogicIn(mode);
unsigned short mode;    /* input lines to be enabled */
```

Description: Enables or disables Logic Unit input line. Enabling a Logic Unit input causes the selected signal to be routed through the Logic Unit.

The lower 7 bits of `mode` correspond to the Logic Unit's input lines 0-6. If bit x of `mode` is 1, input line x will be enabled. If bit x of `mode` is 0, input line x will be disabled.

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

`ERR_PARA`	Parameter error
`ERR_HW`	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFEnLogicOut

ATFEnLogicOut

Purpose: Enables or disables Logic Unit output.

Syntax:

```
short far pascal ATFEnLogicOut(mode);
unsigned short mode;     /* output mode */
```

Description: Enables or disables Logic Unit output. The Logic Unit output is the logic OR of all the enabled inputs. The **ATFSetLogicIn** and **ATFSetLogicOut** functions are used to select other logic functions such as AND, NOR and NAND.

The following constants are defined in the **atf.h** include file:

Legal values for `mode` are:

`ON`	Enable output signal
`OFF`	Disable output signal

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

`ERR_PARA`	Parameter error
`ERR_HW`	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFEnOutBNC

ATFEnOutBNC

Purpose: Enables or disables Output BNCs.

Syntax:

```
short far pascal ATFEnOutBNC(bnc, mode);
unsigned short bnc;      /* Output BNC */
unsigned short mode;     /* enable or disable */
```

Description: Enables or disables Output BNCs. Enabling an output BNC causes the connected signal to be output.

The following constants are defined in the **atf.h** include file.

Legal values for `bnc` are:

`OBNCA`	Output BNC A
`OBNCB`	Output BNC B

Legal values for `mode` are:

`ON`	Enable Output BNC
`OFF`	Disable Output BNC

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

`ERR_PARA`	Parameter error
`ERR_HW`	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFEnProtoU — ATFEnableProtocolU

Purpose: Enables or disables Protocol Unit.

Syntax:

```
short far pascal ATFEnProtoU(unit, mode);
unsigned short unit;     /* Protocol Unit */
unsigned short mode;     /* enable or disable */
```

Description: Enables or disables Protocol Unit. Enabling a Protocol Unit allows the selected protocol to be generated upon receipt of an input signal.

The following constants are defined in the **atf.h** include file.

Legal values for `unit` are:

PROTO1	Protocol Unit 1
PROTO2	Protocol Unit 2

Legal values for `mode` are:

ON	Enable Protocol Unit
OFF	Disable Protocol Unit

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

ERR_PARA	Parameter error
ERR_HW	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFOutBNCRt

ATFOutBNCRt

Purpose: Routes a signal from one of the ITB lines to Output BNC.

Syntax:

```
short far pascal ATFOutBNCRt(bnc, itb, polarity);
unsigned short bnc;              /* output BNC */
unsigned short itb;              /* ITB line */
unsigned short polarity;         /* signal polarity */
```

Description: Routes a signal from one of the ITB lines to Output BNC and selects the polarity of the output signal.

The following constants are defined in the **atf.h** include file.

Legal values for `bnc` are:

`OBNCA`	Output BNC A
`OBNCB`	Output BNC B

Legal values for `itb` are:

`ITB0...7`	ITB lines 0 - 7

Legal values for `polarity` are:

`POS`	Non-inverted output signal
`NEGA`	Inverted output signal

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

`ERR_PARA`	Parameter error
`ERR_HW`	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFOutBNCBStat

ATFOutBNCBStat

Purpose: Reads the current Output BNC B status.

Syntax: `short far pascal ATFOutBNCBStat( );`

Description: This function reads the current status of Output BNC B.

Return Value: If the output voltage is high, it returns a 1.
If the output voltage is low, it returns a 0.

Errors:

ERR_HW	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFPowerTempStat

ATFPowerTempStat

Purpose: Measures power supply voltage and the board temperature inside the VX5530/VX5535.

Syntax:

```
short far pascal ATFPowerTempStat(selection, result);
unsigned short selection; /* supply/temperature selection */
float far *result;        /* current reading */
```

Description: Measures the current power supply voltages and current board temperature inside the VX5530/VX5535.

The power supply voltage reported in `result` can be one of the following:

+24V, +12V, +5V, –24V, –12V, –5.2V, –2V.

The following constants are defined in the **atf.h** include file.

Legal values for `selection` are:

`P24V`	+24 V power supply
`P12V`	+12 V power supply
`P5V`	+5 V power supply
`N24V`	–24 V power supply
`N12V`	–12 V power supply
`N5PT2V`	–5.2 V power supply
`N2V`	–2 V power supply
`TEMPERATURE`	board temperature in Celsius

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

`ERR_PARA`	Parameter error
`ERR_HW`	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFSelECLTRG

ATFSelECLTRG

Purpose: Selects ECL Trigger line.

Syntax:

```
short far pascal ATFSelECLTRG(unit, ecltrg);
unsigned short unit;          /* Protocol Unit */
unsigned short ecltrg;        /* ECL Trigger Line */
```

Description: Selects an ECL Trigger line for the Protocol Unit. If the Protocol Unit is set up for Extended Start Stop protocol or SYNC100 protocol, this function is used to select the ECL Trigger line for the protocol.

The following constants are defined in the **atf.h** include file.

Legal values for `unit` are:

PROTO1	Protocol Unit 1
PROTO2	Protocol Unit 2

Legal values for `ecltrg` are:

ECLTRG0	ECL Trigger Line 0
ECLTRG1	ECL Trigger Line 1
ECLTRG2	ECL Trigger Line 2
ECLTRG3	ECL Trigger Line 3
ECLTRG4	ECL Trigger Line 4
ECLTRG5	ECL Trigger Line 5

NOTE

After executing this function, the Protocol Unit has to be enabled again before sending another trigger.

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

ERR_PARA	Parameter error
ERR_HW	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFSelSysClk

ATFSelSysClk

Purpose: Selects internal/external clock.

Syntax:

```
short far pascal ATFSelSysClk(clock);
unsighed short clock;   /* clock source */
```

Description: Selects either internal or external clock as the CLK10 signal.

If internal clock is selected, the CLK10 signal is generated by an internal clock in the VX5530/ VX5535.

If external clock is selected, the CLK10 signal is phase-locked with the signal on ITB6 to maintain the clock rate at 10 MHz.

The following constants are defined in the **atf.h** include file:

Legal values for `clock` are:

`INTERNAL`	Internal clock
`EXTERNAL`	External clock

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

`ERR_PARA`	Parameter error
`ERR_HW`	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFSelTTLTRG

ATFSelTTLTRG

Purpose: Selects TTL Trigger line.

Syntax:

```
short far pascal ATFSelTTLTRG(unit, ttltrg);
unsigned short unit;          /* Protocol Unit */
unsigned short ttltrg;        /* TTL Trigger Line */
```

Description: Selects a TTL Trigger line for the Protocol Unit. If the Protocol Unit is set up for Start Stop or Extended Start Stop protocol, this function is used to select the TTL Trigger line for the protocol.

The following constants are defined in the **atf.h** include file.

Legal values for `unit` are:

PROTO1	Protocol Unit 1
PROTO2	Protocol Unit 2

Legal values for `ttltrg` are:

TTLTRG0	TTL Trigger Line 0
TTLTRG1	TTL Trigger Line 1
TTLTRG2	TTL Trigger Line 2
TTLTRG3	TTL Trigger Line 3
TTLTRG4	TTL Trigger Line 4
TTLTRG5	TTL Trigger Line 5
TTLTRG6	TTL Trigger Line 6
TTLTRG7	TTL Trigger Line 7

NOTE

After executing this function, the Protocol Unit has to be enabled again before sending another trigger.

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

ERR_PARA	Parameter error
ERR_HW	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFSendTrg ATFSendTrg

Purpose: Sends internal μP trigger.

Syntax:

```
short far pascal ATFSendTrg(unit, command);
unsigned short unit;            /* Protocol Unit */
unsigned short command;         /* trigger command */
```

Description: Sends internal μP trigger command to a Protocol Unit. Before executing this function, the Protocol Unit must be set up and enabled.

The following constants are defined in the **atf.h** include file.

Legal values for `unit` are:

PROTO1	Protocol Unit 1
PROTO2	Protocol Unit 2
PROTOBOTH	Protocol Units 1 and 2

Legal values for `command` are:

STARTTRG	START command
STOPTRG	STOP command

NOTE

It is important to use trigger commands in pairs: if you issue a STOP command, it must be followed by a START command, and vice versa.

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

ERR_PARA	Parameter error
ERR_HW	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFSetDelayU

ATFSetDelayU

Purpose: Sets up Delay Unit.

Syntax:

```
short far pascal ATFSetDelayU(delayunit, delaytime);
unsigned short delayunit;      /* delay unit */
unsigned short delaytime;      /* delay time */
```

Description: Sets up Delay Unit. The delay time ranges from 0 ns to 15 ns.

The following constants are defined in the **atf.h** include file.

Legal values for `delayunit` are:

DELAY1	Delay Unit 1
DELAY2	Delay Unit 2

Legal values for `delaytime` are:

DELAY0NS	Delay 0 ns
DELAY1NS	Delay 1 ns
DELAY2NS	Delay 2 ns
DELAY3NS	Delay 3 ns
DELAY4NS	Delay 4 ns
DELAY5NS	Delay 5 ns
DELAY6NS	Delay 6 ns
DELAY7NS	Delay 7 ns
DELAY8NS	Delay 8 ns
DELAY9NS	Delay 9 ns
DELAY10NS	Delay 10 ns
DELAY11NS	Delay 11 ns
DELAY12NS	Delay 12 ns
DELAY13NS	Delay 13 ns
DELAY14NS	Delay 14 ns
DELAY15NS	Delay 15 ns

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

ERR_PARA	Parameter error
ERR_HW	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFSetLogicIn

ATFSetLogicIn

Purpose: Sets up Logic Unit input.

Syntax:

```
short far pascal ATFSetLogicIn(inputline, mode);
unsigned short inputline;      /* Logic Unit input line */
unsigned short mode;           /* input line setup */
```

Description: Sets up Logic Unit input line. The Logic Unit input lines correspond to ITB lines. The Logic Unit input can be set up for level or edge sensitive and for non-inverted or inverted signals. Combining **ATFSetLogicIn** with **ATFSetLogicOut** allows the Logic Unit to perform various logic functions such as AND, NOR and NAND.

The following constants are defined in the **atf.h** include file.

Legal values for `inputline` are:

LOGICI0	Logic unit input 0—ITB0
LOGICI1	Logic unit input 1—ITB1
LOGICI2	Logic unit input 2—ITB2
LOGICI3	Logic unit input 3—ITB3
LOGICI4	Logic unit input 4—ITB4
LOGICI5	Logic unit input 5—ITB5
LOGICI6	Logic unit input 6—ITB6

Legal values for `mode` are:

LEVEL_POS	Level sensitive and non-inverted
LEVEL_NEG	Level sensitive and inverted
EDGE_POS	Edge sensitive and non-inverted
EDGE_NEG	Edge sensitive and inverted

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

ERR_PARA	Parameter error
ERR_HW	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFSetLogicOut

ATFSetLogicOut

Purpose: Sets up Logic Unit output.

Syntax:

```
short far pascal ATFSetLogicOut(mode);
unsigned short mode;     /* output mode */
```

Description: Sets up Logic Unit output mode. The output of the Logic Unit can be set to non-inverted or inverted mode. Combining **ATFSetLogicIn** with **ATFSetLogicOut** allows the Logic Unit to perform various logic functions such as AND, NOR and NAND.

The following constants are defined in the **atf.h** include file.

Legal values for `mode` are:

`POS`	Non-inverted output signal
`NEGA`	Inverted output signal

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

`ERR_PARA`	Parameter error
`ERR_HW`	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFSetProtoU

ATFSetProtoU

Purpose: Sets up Protocol Unit.

Syntax:

```
short far pascal ATFSetProtoU(unit, trgsrc, protocol);
unsigned short unit;          /* protocol unit */
unsigned short trgsrc;        /* trigger source */
unsigned short protocol;      /* trigger protocol */
```

Description: Sets up Protocol Unit. There are two Protocol Units in the VX5530/VX5535. Both of them have the same functionality. The trigger source for the Protocol Unit can be either from external or internal μP trigger.

For external trigger, ITB4 is used to trigger Protocol Unit 1 and ITB5 is used to trigger Protocol Unit 2.

For internal μP trigger, **ATFSendTrg** is used to send the trigger command.

Start Stop, Extended Start Stop or SYNC100 protocol can be selected for the Protocol Unit. In that case, **ATFSelECLTRG** or **ATFSelTTLTRG** functions are used to select the ECL Trigger line or the TTL Trigger line to be used in the protocol.

The following constants are defined in the **atf.h** include file.

Legal values for `unit` are:

`PROTO1`	Protocol Unit 1
`PROTO2`	Protocol Unit 2

Legal values for `trgsrc` are:

`INTERNAL`	Internal μP trigger
`EXTERNAL`	External trigger

Legal values for `protocol` are:

`STARTSTOP`	Start Stop protocol
`ESTARTSTOP`	Extended Start Stop protocol
`SYNC100`	SYNC100 protocol

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

`ERR_PARA`	Parameter error
`ERR_HW`	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFSTARRoute

ATFSTARRoute

Purpose: Routes a signal through a STARbus routing unit.

Syntax:

```
short far pascal ATFSTARRoute(unit, source, destination);
unsigned short unit;              /* STARbus routing unit */
unsigned short source;            /* STARbus routing unit source
                                  channel */
unsigned short destination;       /* STARbus routing unit
                                  destination channel */
```

Description: Routes a signal through a STARbus routing unit.

`unit` specifies the STARbus routing unit to use. See legal values below. STARbus routing units 1 and 2 are not supported for Option 01 hardware.

`source` specifies the STARbus routing unit input channel. See legal values below.

`destination` specifies the STARbus routing unit channels the signal is routed to. Multiple destination channels can be selected at the same time by ORing the channel constants.

The following constants are defined in the **atf.h** include file.

Legal values for `unit` are:

SR0	STARbus routing unit 0
SR1	STARbus routing unit 1
SR2	STARbus routing unit 2

Legal values for `source` are:

ITB0-7	ITB lines 0..7, available for all three Routing Units
STARX2-5	STARX 2..5 (STARbus routing unit 0)
STARY2-5	STARY 2..5 (STARbus routing unit 0)
STARX6-9	STARX 6..9 (STARbus routing unit 1)
STARY6-9	STARY 6..9 (STARbus routing unit 1)
STARX1,STARX10-12	STARX 1, STARX 10..12 (STARbus routing unit 2)
STARY1,STARY10-12	STARY 1, STARY 10..12 (STARbus routing unit 2)
SRUPROC	STAR Route microprocessor control signal
SRCLK10	Clock 10 signal

The STAR Route microprocessor control signal can be routed to the destination channel only through STARbus routing unit 1 or STARbus routing 2. The `CLK10` signal can be routed to the destination channel only through STARbus routing unit 0.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Legal values for `destination` are:

`ITB0-7`	ITB lines 0..7, available for all three Routing Units
`STARX2-5`	STARX 2..5 (STARbus routing unit 0)
`STARY2-5`	STARY 2..5 (STARbus routing unit 0)
`STARX6-9`	STARX 6..9 (STARbus routing unit 1)
`STARY6-9`	STARY 6..9 (STARbus routing unit 1)
`STARX1,STARX10-12`	STARX 1, STARX 10..12 (STARbus routing unit 2)
`STARY1,STARY10-12`	STARY 1, STARY 10..12 (STARbus routing unit 2)

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

`ERR_PARA`	Parameter error
`ERR_HW`	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ATFSTARuproc

ATFSTARuproc

Purpose: Sets STARbus routing unit microprocessor control signal.

Syntax:

```
short far pascal ATFSTARuproc(unit, mode);
unsigned short unit;     /* STARbus routing unit */
unsigned short mode;     /* microprocessor control signal */
```

Description: This function sets up the microprocessor control signal to be routed by a STARbus routing unit. The STARbus routing unit can route an microprocessor control signal (controlled by the microprocessor) to any other channel by using the function **ATFSTARRoute** . The microprocessor control signal can be set to high or low value by the **ATFSTARuproc** function. This function is not supported for Option 01 hardware.

The following constants are defined in the **atf.h** include file.

Legal values for `unit` are:

SR1	STARbus routing unit 1
SR2	STARbus routing unit 2

Legal values for `mode` are:

POS	High voltage level
NEGA	Low voltage level

NOTE

This function may result in a different signal being routed to Output BNC B. Do not use Output BNC B and this function at the same time.

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns one of the following:

ERR_PARA	Parameter error
ERR_HW	Hardware not available

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Advanced Trigger Function Examples

Example 1: Routes a signal from Input BNC A to Output BNC B, and inverts the output signal.

Input BNC A is connected to ITB6.

```
ATFEnInBNC(IBNCA, ON);          /* enable input BNC A */
ATFOutBNCRt(OBNCB, ITB6, NEGA); /* route from ITB6 to Output BNC B
                                   and invert the output signal */
ATFEnOutBNC(OBNCB, ON);         /* enable output BNC B */
```

Example 2: Do a logic OR with the Input BNC A signal and the Input BNC B signal, then output the resulting signal to Output BNC A.

Input BNC A is connected to ITB6 and Input BNC B is connected to ITB4.

```
ATFEnInBNC(IBNCA, ON);              /* enable input BNC A */
ATFEnInBNC(IBNCB, ON);              /* enable input BNC B */
ATFSetLogicIn(LOGICI4, LEVEL_POS);  /* set up logic input 4 (Input BNC B)
                                       for level sensitive and non-inverted */
ATFSetLogicIn(LOGICI6, LEVEL_POS);  /* set up logic input 6 (Input BNC A)
                                       for level sensitive and non-inverted */
ATFEnLogicIn(0x50);                 /* Logic Unit input lines 4 and 6
                                       are used */
ATFEnLogicOut(ON);                  /* enable logic output */
ATFOutBNCRt(OBNCA, ITB7, POS);      /* route from ITB7 to Output BNC A */
ATFEnOutBNC(OBNCA, ON);             /* enable Output BNC A */
```

Example 3: Routes a signal from Input BNC A to STARX2 and STARY3.

Input BNC A is connected to ITB6.

```
ATFEnInBNC(IBNCA, ON);                  /* enable input BNC A */
ATFSTARRoute(SR0, ITB6, STARX2|STARY3); /* In STARbus routing unit 0,
                                           signal from ITB6 is routed to
                                           STARY3 and STARX2 */
ATFEnSRRec(SR0, ITB6);                  /* enable ITB6 receiver */
ATFEnSRDrv(SR0, STARX2|STARY3);         /* enable STARX2 + STARY3 driver */
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example 4: Routes a signal from STARX7 to STARY3, STARX9, and STARY12.

ITB3 is selected to route the signal from STARbus Routing Unit 1 to STARbus Routing Units 0 and 2. Signal from STARX7 is routed to ITB3 and STARX9, then from ITB3 to STARY3 and STARY12.

```
ATFSTARRoute(SR1, STARX7, ITB3|STARX9); /* route from STARX7 to ITB3 and
                                           STARX9 */
ATFEnSRRec(SR1, STARX7);                /* enable STARX7 receiver */
ATFEnSRDrv(SR1, ITB3|STARX9);           /* enable ITB3 and STARX9 driver */

ATFSTARRoute(SR0, ITB3, STARY3);        /* route from ITB3 to STARY3 */
ATFEnSRRec(SR0, ITB3);                  /* enable ITB3 receiver */
ATFEnSRDrv(SR0, STARY3);                /* enable STARY3 driver */

ATFSTARRoute(SR2, ITB3, STARY12);       /* route from ITB3 to STARY12 */
ATFEnSRRec(SR2, ITB3);                  /* enable ITB3 receiver */
ATFEnSRDrv(SR2, STARY12);               /* enable STARY12 driver */
```

Example 5: Routes a signal from Input BNC A to Delay Unit 1. Delay Unit 1 is set up to delay 10 ns and the output of Delay Unit 1 is routed to Output BNC B.

Since Input BNC A uses ITB6 as input and Delay Unit 1 uses ITB0 as input, the signal from ITB6 must route through one of the STARbus Routing Units to ITB0. In this example, STARbus Routing Unit 0 is chosen.

```
ATFEnInBNC(IBNCA, ON);              /* enable Input BNC A */

ATFSetDelayU(DELAY1, DELAY10NS);    /* signal delay 10 ns through
                                       Delay Unit 1 */
ATFEnDelayU(DELAY1, ON);            /* enable Delay Unit 1 */

ATFSTARRoute(SR0, ITB6, ITB0);      /* route from ITB6 to ITB0 in
                                       STARbus routing unit 0 */
ATFEnSRRec(SR0, ITB6);              /* enable ITB6 receiver */
ATFEnSRDrv(SR0, ITB0);              /* enable ITB0 driver */

ATFOutBNCRt(OBNCB, ITB1, POS);      /* signal from ITB1 to Output BNC B */
ATFEnOutBNC(OBNCB, ON);             /* enable Output BNC B */
```

Example 6: Routes the internal CLKL10 signal to Output BNC A.

ITB4 is chosen to route the signal.

```
ATFSelSysClk(INTERNAL);             /* select internal CLK10 signal */
ATFSTARRoute(SR0, SRCLK10, ITB4);   /* route from CLK10 to ITB4 */
ATFEnSRDrv(SR0, ITB4);              /* enable ITB4 driver */

ATFOutBNCRt(OBNCA, ITB4, POS);      /* route from ITB4 to Output BNC A */
ATFEnOutBNC(OBNCA, ON);             /* enable Output BNC A */
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example 7: Do a logic OR with the STARbus Routing Unit 1 microprocessor control and STARX4, and route the resulting signal to STARX10.

ITB3 is chosen to route STARbus Routing Unit 1 microprocessor control signal. ITB4 is chosen to route the signal from STARX4 to the Logic Unit. The **ATFSTARuproc** function is used to set the microprocessor control signal to High.

```
ATFSTARRoute(SR1, SRUPROC, ITB3);     /* route from STARbus Routing Unit 1
                                      microprocessor control to ITB3 */
ATFEnSRDrv(SR1, ITB3);                /* enable ITB3 driver */

ATFSTARRoute(SR0, STARX4, ITB4);      /* route from STARX4 to ITB4 */
ATFEnSRRec(SR0, STARX4);              /* enable STARX4 receiver */
ATFEnSRDrv(SR0, ITB4);                /* enable ITB4 driver */

ATFSetLogicIn(LOGICI3, LEVEL_POS);    /* set up Logic Unit 3 */
ATFSetLogicIn(LOGICI4, LEVEL_POS);    /* set up Logic Unit 4 */
ATFEnLogicIn(0x18);                   /* enable logic input 3 and 4 */
ATFEnLogicOut(ON);                    /* enable logic output */

ATFSTARRoute(SR2, ITB7, STARX10);     /* route from ITB7 to STARX10 */
ATFEnSRRec(SR2, ITB7);                /* enable ITB7 receiver */
ATFEnSRDrv(SR2, STARX10);             /* enable STARX10 driver */

ATFSTARuproc(SR1, POS);               /* set microprocessor control
                                      signal HIGH */
```

Example 8: Send Extended Start Stop protocol from Protocol Unit 1 using ECL Trigger line 2 and TTL Trigger line 4 with the processor controlling the trigger command. After the setup, a START command and a STOP command are sent.

```
ATFSetProtoU(PROTO1, INTERNAL, ESTARTSTOP); /* Protocol Unit 1 is set up
                                            for internal trigger and
                                            Extended Start Stop protocol */
ATFSelECLTRG(PROTO1, ECLTRG2);              /* Protocol Unit 1 uses
                                            ECL Trigger line 2 */
ATFSelTTLTRG(PROTO1, TTLTRG4);              /* Protocol Unit 1 uses
                                            TTL Trigger line 4 */
ATFEnProtoU(PROTO1, ON);                    /* enable Protocol Unit 1 */

ATFSendTrg(PROTO1, STARTTRG);               /* send START command */
ATFSendTrg(PROTO1, STOPTRG);                /* send STOP command */
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Example 9: Sends SYNC100 protocol from Protocol Unit 1 using ECL Trigger line 5. The trigger command comes from Input BNC B.

Both Input BNC B and Protocol Unit 1 are connected to ITB4.

```
ATFSetProtoU(PROTO1, EXTERNAL, SYNC100);     /* Protocol Unit 1 is set up
                                                for external trigger and
                                                SYNC100 protocol */
ATFSelECLTRG(PROTO1, ECLTRG5);               /* Protocol Unit 1 uses ECL
                                                Trigger line 5 */
ATFEnProtoU(PROTO1, ON);                     /* enable Protocol Unit 1 */
ATFEnInBNC(IBNCB, ON);                       /* enable Input BNC B */
```

Example 10: Measures current +24 V, + 5 V power supply voltages and current board temperature.

```
ATFPowerTempStat(P24V, (float far *) &result);   /* measure +24V
                                                    voltage */
ATFPowerTempStat(P5V, (float far *) &result);    /* measure +5V voltage */
ATFPowerTempStat(TEMPERATURE, (float far *) &result);   /* measure
                                                           temperature */
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Appendix A
I/O Control Commands

This Appendix contains a description of I/O Control commands used by the GPIB and VXIbus bridges. The define files containing these commands are **gpibrdg.h** and **vxicmd.h**, respectively.

GPIBRDG Control Commands

These commands allow the user to alter temporarily the default communication attributes for a device or to perform GPIB functions (such as Interface Clear). The commands are described in Table A-1.

Note that there is no restriction on the number of devices specified by the configurator. However, the bridge must keep track of changes that are programmably made to device attributes (such as timeout). The limit to the number of attribute default changes is fifty devices.

It is recommended that the users of these commands understand how to control the GPIB at a low level before attempting to use the Low Level Bus control functions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table A-1 GPIBRDG Control Commands

Command	Return Value	Description
GPIBINITIALIZE	0	Initializes the GPIB bridge. This removes any changes to default device characteristics that may have been changed by the GPIBRDG control commands (for example, TIMEOUT).
GPIBVERSION	Upper byte = major version, lower byte = minor version	Returns the software revision number of the GPIBRDG.
GPIBFUNCTIONSERIALPOLL	Device status byte	Performs a serial poll on the selected device and returns the device status byte.
GPIBFUNCTIONPARALLELPOLL	Result of parallel poll	Performs a parallel poll and returns the result. Use as follows: GPIBFUNCTIONPARALLELPOLL |0x00YY0000; /* YY = hex byte code */
GPIBFUNCTIONPPOLLCONFIG	0	Allows the device to be configured for a parallel poll.
GPIBFUNCTIONGOTOLOCAL	0	Allows control to be temporarily returned to the device front panel.
GPIBFUNCTIONDEVICECLEAR	0	Allows the device to be cleared.
GPIBFUNCTIONTRIGGER	0	Sends the Group Execute Trigger (GET) command to the device. All devices may be triggered using the GPIBCOMMANDGET command.
GPIBFUNCTIONREADBYTE	0	Allows a single byte to be read from the device.
GPIBFUNCTIONWRITEBYTE	0	Allows a single byte to be written to the device.
GPIBMODEEOIWRITEON	0	Specifies that the EOI signal will be asserted at the end of the message.
GPIBMODEEOIWRITEOFF	0	Specifies that the EOI signal will NOT be asserted at the end of the message.
GPIBMODEEOSENDREAD	0	Specifies that a received message will be terminated when an EOS condition is detected.
GPIBMODEEOSREADMORE	0	Specifies that a received message will NOT be terminated when an EOS condition is detected.
GPIBMODEEOSEOION	0	Specifies that EOI will be asserted with EOS.
GPIBMODEEOSEOIOFF	0	Specifies that EOI will NOT be asserted with EOS.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table A-1 GPIBRDG Control Commands (Cont)

Command	Return Value	Description
GPIBMODEEOS8BIT	0	Specifies that an eight bit compare will be used to detect EOS.
GPIBMODEEOS7BIT	0	Specifies that a seven bit compare will be used to detect EOS.
GPIBMODEEOSBYTEMASK	0	This command is the mask for specifying the EOS byte. Use as follows: GPIBMODEEOSBYTEMASK|0x00YY0000; /* YY = hex byte code */
GPIBTIMEOUTOFF	0	Sets the interface timeout for current device to infinity.
GPIBTIMEOUT10US GPIBTIMEOUT30US GPIBTIMEOUT100US GPIBTIMEOUT300US GPIBTIMEOUT1MS GPIBTIMEOUT3MS GPIBTIMEOUT10MS GPIBTIMEOUT30MS GPIBTIMEOUT100MS GPIBTIMEOUT300MS GPIBTIMEOUT1S GPIBTIMEOUT3S GPIBTIMEOUT10S GPIBTIMEOUT30S GPIBTIMEOUT100S GPIBTIMEOUT300S GPIBTIMEOUT1000S	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Sets the timeout for current device to the time (approx) specified.
GPIBCOMMANDMASK	0	MASK is used to specify the low level GPIB message. Use as follows: GPIBCOMMANDMASK|0x00YY0000; /* YY = low level interface message */

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table A-1 GPIBRDG Control Commands (Cont)

Command	Return Value	Description
		This command is a pre-coded version of the GPIBCOMMANDMASK that sends the message specified as follows:
GPIBCOMMANDGTL	0	Go To Local
GPIBCOMMANDSDC	0	Selected Device Clear
GPIBCOMMANDPPC	0	Parallel Poll Configure
GPIBCOMMANDGET	0	Group Execute Trigger
GPIBCOMMANDTCT	0	Take ConTrol
GPIBCOMMANDLLO	0	Local LockOut
GPIBCOMMANDDCL	0	Device CLear
GPIBCOMMANDPPU	0	Parallel Poll Unconfigure
GPIBCOMMANDSPE	0	Serial Poll Enable
GPIBCOMMANDSPD	0	Serial Poll Disable
GPIBCOMMANDUNT	0	UNTalk
GPIBCOMMANDUNL	0	UNListen
GPIBCOMMANDLA0	0	Listen primary Address 0
GPIBCOMMANDTA0	0	Talk primary Address 0
GPIBCOMMANDSA0	0	Secondary Address 0
GPIBCONTROLIFC	0	Pulses the Interface Clear bus control signal.
GPIBCONTROLDMAON	0	Turns DMA on.
GPIBCONTROLDMAOFF	0	Turns DMA off.
GPIBRENON	0	Asserts the Remote Enable bus control line. The REN line tells all bus connected devices that they are under computer control, as opposed to front panel control. (Use GPIBFUNCTIONGOTOLOCAL to obtain temporary front panel control.)
GPIBRENOFF	0	Unasserts the Remote Enable bus control line. When the REN line is unasserted, front panel control is allowed in the device.
GPIBSTATUS	Interface status	Returns the status of the GPIB interface. For more information on the status word assignments, see Table A-2.
GPIBSTATUSSRQ	0 = SRQ unasserted, 1 = SRQ asserted	Returns the status of the GPIB interface signal called SRQ. The SRQ is an interrupt signal controlled by the instruments to tell the controller they need attention. The SRQ is usually unasserted when the device asserting SRQ has been serially polled.
GPIBSTATUSEND	0 = not at end, 1 = at end of message	Returns a value indicating whether the end of the message has been reached.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table A-1 GPIBRDG Control Commands (Cont)

Command	Return Value	Description
GPIBSTATUSERROR	Error	Returns the last interface error. This must be used in conjunction with the GPIBSTATUS command which indicates validity. For more information, refer to Table A-3.
GPIBSTATUSCOUNT	Number-of-bytes-transferred	Returns the count in number-of-bytes-transferred, as a result of the last I/O function performed.
GPIBCURRENTTIMEOUT	Timeout code	Returns the current timeout code for the device. Current settings show attributes that may have been changed from the default settings via the control commands.
GPIBCURRENTEOSBYTE	EOS byte code	Returns the current EOS byte for the device. Current settings show attributes that may have been changed from the default settings via the control commands.
GPIBCURRENTEOSMODE	EOS mode code	Returns the current EOS mode for the device. Current settings show attributes that may have been changed from the default settings via the control commands. Mode code: Bit 0 = (1) terminate read on EOS Mode code: Bit 1 = (1) Set EOI with EOS on write Mode code: Bit 2 = (1) Eight bit compare (0) seven bit compare
GPIBCURRENTEOIMODE	0 = EOI off 1 = EOI on	Returns the current EOI mode for the device. Current settings show attributes that may have been changed from the default settings via the control commands.
GPIBCURRENTPRIMARY	Primary address	Returns the current primary address for the device. Current settings show attributes that may have been changed from the default settings via the control commands.
GPIBCURRENTSECONDARY	Secondary address	Returns the current secondary address for the device. Current settings show attributes that may have been changed from the default settings via the control commands.
GPIBDEFAULTTIMEOUT	Timeout code	Returns the default timeout code for the device. See Table A-4 for a list of default timeout codes. Default settings come from the GPIB Configurator.
GPIBDEFAULTEOSBYTE	EOS byte code	Returns the default EOS byte for the device.Default settings come from the GPIB Configurator.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table A-1 GPIBRDG Control Commands (Cont)

Command	Return Value	Description
GPIBDEFAULTEOSMODE	EOS mode code	Returns the EOS mode settings for the device. Default settings come from the GPIB Configurator. Mode code: Bit 0 = (1) terminate read on EOS Mode code: Bit 1 = (1) Set EOI with EOS on write Mode code: Bit 2 = (1) Eight bit compare (0) seven bit compare
GPIBDEFAULTEOIMODE	0 = EOI off 1 = EOI on	Returns the default EOI mode for the device. Default settings come from the GPIB Configurator.
GPIBDEFAULTPRIMARY	Primary address	Returns the default primary address for the device. Default settings come from the GPIB Configurator.
GPIBDEFAULTSECONDARY	Secondary address	Returns the default secondary address for the device. Default settings come from the GPIB Configurator.

Table A-2 Status Word Layout

Bit Position	Hex Value	Description
15	8000	GPIB error.
14	4000	Time limit exceeded.
13	2000	GPIB-PC detected END or EOS.
12	1000	SRQ on.
11	800	Device requesting service.
8	100	I/O completed.
7	80	GPIB-PC is in lockout state.
6	40	GPIB-PC is in remote state.
5	20	GPIB-PC is controller-in-charge.
4	10	Attention is asserted.
3	8	GPIB-PC is talker.
2	4	GPIB-PC is listener.
1	2	GPIB-PC is in device trigger state.
0	1	GPIB-PC is in device clear state.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table A-3 GPIB Error Codes

Decimal Value	Explanation
0	DOS error.
1	Function requires GPIB-PC to be controller-in-charge.
2	No listener on write function.
3	GPIB-PC not addressed correctly.
4	Invalid argument to function call.
5	GPIB-PC not system controller as required.
6	I/O operation aborted.
7	Non-existent GPIB-PC board.
10	I/O started before previous operation completed.
11	No capability for operation.
12	File system error.
14	Command error during device call.
15	Serial poll status byte lost.
16	SRQ locked in ON position.

Table A-4 GPIB Timeout Codes

Actual Value	Minimum Timeout
0	disabled
1	10 μs
2	30 μs
3	100 μs
4	300 μs
5	1 ms
6	3 ms
7	10 ms
8	30 ms
9	100 ms
10	300 ms
11	1 s
12	3 s
13	10 s
14	30 s
15	100 s
16	300 s
17	1000 s

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

VXIbus Bridge Control Commands

NOTE

VXIbus communications can be changed between Word Serial protocol and Shared-Memory protocol with the IOUPGRADE and IODOWNGRADE commands.

VXIbus Word Serial Protocol

Table A-5 shows the I/O control commands for the VXIbus Word Serial protocol.

Table A-5 VXIbus Word Serial Control Commands

Command	Return Value	Description
TIMEOUT100MS **TIMEOUT1S** **TIMEOUT10S** **TIMEOUT100S** **TIMEOUT1000S**	No response	Sets timeout value to time specified.
BRIDGEVERSION	Lower byte = minor version, upper byte = major version	Returns the software version number.
DEVICECLEAR	No response	Sends Word Serial Clear command to device.
DEVICETRIGGER	No response	Sends Word Serial Trigger command to device.
DEVICESTATUS	Device status byte	Sends Word Serial Read STB command to device.
WSCOMMAND	Depends on command sent	Sends Word Serial command to device. The first word of the argument contains the actual Word Serial command.
LWSCOMMAND	Depends on command sent	Sends Long Word Serial command to device. The first long word of the argument contains the actual Long Word Serial command.
ELSCOMMAND	Depends on command sent	Sends Extended Long Word Serial command to device. The first two words of the argument contain the actual Extended Long Word Serial command.
IOUPGRADE	0 = success 1 = failure	Change the VXI communication protocol to Shared Memory protocol.

Since all the VXIbus Word Serial I/O control calls return a 16 bit value, the upper word of the response value should be ignored.

For example, to send the Word Serial command "Abort Normal Operation" to device "dvm1", the appropriate lines of code would be:

```
unsigned short wscmd;
unsigned long response;
wscmd = 0xD8FF;
mpctrl("dvm1", WSCOMMAND, &wscmd, &response);
```

To set the timeout value of device "dvm1" to 1 second, the code would be:

```
mpctrl("dvm1", TIMEOUT1S, NULL, NULL);
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Shared-Memory Protocol

Table A-6 shows the five special control services associated with the VXIbus Shared-Memory Protocol.

Table A-6 VXIbus Shared-Memory Protocol Commands

Command	Return Value	Description
IOEOM	No response	Sets a binary switch inside the Shared-Memory bridge. If this switch is set, the bridge will transmit an End-Of-Message token after each bridge write operation. If the switch is cleared, no EOM will be transmitted. This feature allows a 488.2 message unit to be transmitted in several blocks if memory limitations are a consideration. By default, the IOEOM switch is cleared.
IOEOMQRY	Status of the IOEOM switch	Using this function, the programmer can determine if the IOEOM switch is set or cleared, indicating whether or not EOM tokens are being automatically appended by the Shared-Memory bridge.
IOFORMAT	No response	Selects the 32-bit value which will be placed in the Shared-Memory Protocol buffer header format fields. The default value used by the bridge indicates that 488.2 data is carried in the buffers. (Use of the buffer header format field is still undergoing standardization by the Consortium; this control service is intended to allow access to the format field should this become necessary as Shared-Memory Protocol evolves.)
IOTIMEOUT	No response	Specifies how long, in milliseconds, the bridge should wait to complete an I/O transaction. The bridge defaults to 2 seconds.
IOCLEAR	No response	Executes the Shared-Memory Protocol's channel clear protocol—used to clear a Shared-Memory channel that has become overloaded with data. (In normal operation, this control service isn't used.)
IODOWNGRADE	0 = success 1 = failure	Change the VXI communication protocol to Word Serial protocol.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Appendix B
Writing BusManager Applications

Introduction

This section describes how to write drivers using the BusManager.

The Microsoft® C and QuickC TekVICS interface provides access to all BusManager functions. The reader is expected to have some experience writing drivers and interrupt code, and some familiarity with the Microsoft® C version 5.0 (or later) compiler, linker, and (where necessary) assembler and QuickC.

Two categories of VMEbus/VXIbus device drivers are:

1) Program-dedicated drivers. These are drivers that are a part of the program. Typically these drivers are a set of functions that are called by the main part of the code to perform various operations. Most drivers fall into this category.

2) Resident drivers. These are drivers that are loaded at boot time. A resident driver is usually built as a DOS driver and specified for loading in **\config.sys**. A resident driver can also be built as a TSR (terminate-and-stay-resident program) and specified for loading in **\autoexec.bat**.

Program-dedicated drivers have a totally flexible applications interface—arbitrary calls may be easily added. They are also straightforward to implement; however, control of a device is limited to when that program is running.

Resident drivers can make a device accessible to all programs by designating it to be a DOS device, or by defining a service accessible via an interrupt. Writing a resident driver is much more complex than creating program-dedicated drivers. Resident drivers are typically implemented at least partially in assembly language, and often require the creation of an interface library to access device services from higher-level languages.

Using the BusManager Interface

To use TekVICS functions in a driver, include the appropriate TekVICS header files in the modules in which the functions are used, and link your driver object files with the **\tekvics\lib\vcmscx.lib library.**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Bus Window

The primary means of accessing a device is via its control and status registers. These registers are addressable locations, usually in VMEbus A16 space. They are accessed through the VMEbus window. The VMEbus window is a 64KB region of memory within the first megabyte of memory on the VX5530/VX5535 (and therefore easily accessible) which can be mapped to arbitrary sections of the A16, A24, or A32 address spaces on 64KB boundaries. The bus window is only a *master* to the VMEbus—it has no slave address itself and cannot be used as the destination of accesses originated on other boards. This means, for instance, that you can't have a VMEbus device DMA into the bus window.

The mapping of the bus window into the VMEbus address space is controlled by the BusManager, which is installed at boot time and provides the necessary services to use the bus window. The main BusManager functions with regard to the bus window are:

- **MAP BUS.** Set the mapping of the bus window into VMEbus space
- **SET ACCESS MODE.** Set the address modifier (e.g., A16, A24, A32) and the *byte ordering*. The byte ordering may be either Intel-style (aka "little endian") or Motorola-style (aka "big endian").
- **SAVE STATE.** Quickly stash the internal representation of the bus window mapping, address modifier, and byte ordering (collectively known as the *state*) in a location specified by the program.
- **RESTORE STATE.** Quickly restore a previous state from the internal representation saved via SAVE STATE.

Since there may be several drivers accessing the bus window, each mapping it to a different location, care must be taken to save and restore the state used by your driver. The following code fragment demonstrates how this is done.

```
# include       "\tekvics\include\busmgr.h"
    ...
long MyState;           /* my bus window state */

/*
* Device Registers
*/

struct my_device {
    unsigned short      status;     /* status register */
    unsigned short      data[4];    /* data I/O */
};

/* point to device registers */

struct my_device far *MyDev;
    ...
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
/*
 *InitMyDriver -- Initialization entry point for my driver
 */

InitMyDriver()
{
    long old_state;

    /* save state on entry */
    vc_save_state(&old_state);

    /* big endian,A24 space */
    vc_set_acc_mode(BM_MBO | A24N);

    /* get device address: */
    MyDev = (struct my_device far *)
          vc_map_bus(0x400340L);

    /* speed later access*/
    vc_save_state(&MyState);
    ...
    /* restore entry state */
    vc_rest_state(&old_state);
}

/*
*    MyDoOp -- Do an operation on My device
 */

short
MyDoOp(op, arg)
short op;
short arg;
{
    long old_state;
          ...
    /* save entry state */
    vc_save_state(&old_state);

    /* restore device state */
    vc_rest_state(&MyState);

    [manipulate device registers, using MyDev as a pointer
    to the registers]

    /* restore entry state */
    vc_rest_state(&old_state);
}
```

Note how the SAVE STATE and RESTORE STATE operations are used to speed the setup of the bus window.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Interrupts

It is often desirable to use the VMEbus interrupts generated by a device to control its operation. There are 7 VMEbus interrupts, prioritized with 7 as the highest priority and 1 as the lowest priority. While it is possible for several devices to signal via the same interrupt, all the drivers responding to the interrupt must be in the same processor and coordinate amongst themselves—each VMEbus interrupt must be handled by exactly one processor.

When the BusManager detects an interrupt for which it is enabled, it issues a 16-bit Interrupt ACKnowledge (IACK) cycle on the VMEbus bus and gets back an 8- or 16-bit Status/ID response from the interrupting device. The BusManager can't tell the actual size of the response. It is up to the driver to know how many significant bits there are in the response. This Status/ID information is made available to the driver.

The primary BusManager functions for dealing with interrupts include:

- **WAIT FOR INTERRUPT.** Wait until one of a set of interrupts occurs or until the timer expires.
- **SET INTERRUPT HANDLER.** Declare the routine to be called when the specified interrupt occurs.
- **DISABLE VMEbus INTERRUPT.** Mask off an interrupt.
- **ENABLE VMEbus INTERRUPT.** Unmask an interrupt.

Interrupts can be polled, or assigned an interrupt handler. Polling for an interrupt is the easiest way to deal with device interrupts. The polling mechanism provided waits for one of a set of interrupts to occur, or for a timer to expire. The installation and utilization of an interrupt handler requires the setting up of a handler and the establishment of a stack for a specified interrupt.

Polling Interrupts

The easiest way to deal with device interrupts is to use the WAIT FOR INTERRUPT function. The function waits for one of a set of interrupts to occur, or for a timer to expire. No interrupt handler needs to be set up for a specified interrupt, and no stack needs to be established for the interrupt.

The timer used for this and all other timed BusManager operations is the regular DOS 18.2Hz clock. All functions that use the timer enable DOS interrupts internally to allow the clock to run.

The following code fragment shows an example of waiting for an interrupt:

```
long status;              /* returned Status/ID */
    ...
vc_save_state(&old_state);
vc_rest_state(&MyState);
MyDev->data[0] = DATA1;       /* load up data ports */
MyDev->data[1] = DATA2;
MyDev->status |= DEV_GO;      /* turn on go bit */
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
    if (vc_wait_intr((1<Y_INTR), &status, 5*18)
     != (1<<MY_INTR)) {
/*
 * No interrupt!
 */
    ...

    vc_rest_state(&old_state);
    return (FAILURE);
}
/*
 * Process interrupt
 */
    vc_rest_state(&old_state);
    return (SUCCESS);
}
```

Programming hint:

To increase parallelism, consider organizing your design so that instead of issuing a command to the VMEbus device and waiting for it to finish, you wait for the previous device command to complete and then *issue a new command.*

An interrupt may be both "waited for" and also have a handler assigned to it. If there is no handler assigned to an interrupt, that interrupt is not acknowledged until it is "waited for". If there is a handler assigned to an interrupt when the interrupt occurs, the interrupt is acknowledged, the Status/ID and the fact that the interrupt occurred is remembered (recorded internally within the BusManager), and the handler is called passing it the Status/ID. This allows various combinations of drivers to share the same interrupt. By keeping track of interrupts that have occurred before the call to WAIT FOR INTERRUPT, the BusManager assures that no race conditions exist.

As a consequence of having interrupts remembered so that they may be "waited for", drivers that use this function should call WAIT FOR INTERRUPT with a zero timer count during initialization to remove any remembered interrupts.

Disabling and enabling an interrupt (via DISABLE VMEbus INTERRUPT and ENABLE VMEbus INTERRUPT) have no effect on WAIT FOR INTERRUPT. An "awaited" disabled interrupt will be detected as if it were enabled. The difference is that the handler will not be called.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Interrupt Handlers

Interrupt handlers can be implemented in C for the VX5530/VX5535. Polling interrupts is easy for single devices and gives reasonable response time. When several devices are working in parallel or when response time is critical in a multi-tasking environment it is often more appropriate to install interrupt handlers.

The way that the BusManager handles VMEbus interrupts under DOS is that only VMEbus interrupts that have handlers associated with them are unmasked. Interrupts that have no assigned handlers are ignored by the BusManager when they occur on the assumption that some other processor on the VMEbus will handle those interrupts. (The exception to this rule is WAIT FOR INTERRUPT, discussed above). When an interrupt that has a handler assigned to it is detected, the BusManager performs the following operations:

1) Disable DOS interrupts

2) Acknowledge the DOS interrupt (to eliminate race conditions)

3) Determine which VMEbus interrupt was detected

4) Perform the IACK cycle to get the Status/ID and clear the interrupt

5) Save the current state on the BusManager's stack

6) Switch to the handler's stack

7) Perform an ordinary far C call, passing the Status/ID

8) Switch back to the BusManager's stack

9) Scan for another interrupt. (If found, go to step 3)

10) Return to the interrupted DOS routine and reenable DOS interrupts

Each interrupt handler gets its own stack which must have been previously allocated. This stack must have sufficient capacity to store the actual parameters and local variables within the interrupt handler as well as those of subsequent functions which it may call. A stack size of 256 bytes is typical in most applications. This stack is probably not where the C compiler thinks it ought to be, so the interrupt handler must be compiled using the following flags:

/Gs Turn off stack checking. Without this option the handler will immediately think that its stack has overflowed.

/Auxx Tell the compiler that SS != DS, and to reload DS upon entry. The *xx* sets the model, to wit:

Model	Flag
Small	/Ausn
Medium	/Auln
Compact	/Ausf
Large	/Aulf

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

When using the /Auxx flag, pointers that take the address of functions on the stack must be **far** pointers.

There is an alternative to using the /Auxx flag: if the array for the stack is a **near** array (compiling with small or medium model, or explicitly declaring the array as such), the /Auxx flag is unnecessary, since the BusManager sets DS = SS. In other words, if the array used for the stack has the same segment value as your **near** data, then the BusManager will correctly set the data segment register when entering the handler.

The handler function itself must be declared **far**, so that the function entry/exit properly matches the way it is called.

Since Microsoft does not supply libraries that match custom memory models, library functions cannot be called from the handler. Moreover, DOS is not reentrant so no DOS operations can be used while in the handler[1]. The handler *must* return to the BusManager—`setjmp()`/`longjmp()` constructs are not allowed.

However, any BusManager function may be used by the handler. At the very least most handlers will use RESTORE STATE to access their device registers.

The following example shows how to set up an interrupt handler

```
# include"\tekvics\busmgr.h"

# ifndef NULL
# define NULL ((char far *)0)
# endif

# define STKSIZE 256 /* size of intr handler stack */
char MyStack[STKSIZE]; /* interrupt stack */
extern void far MyIntr(); /* interrupt handler */


/*
* Set Up Interrupt Handler
* Don't worry about previous handler for now
*/
    (void)vc_set_intr(MY_INTR, MyIntr,
      &MyStack[STKSIZE], NULL);
```

The handler for interrupt number `MY_INTR` has been set to the function `MyIntr()` and will be called using `MyStack`. Note that `MyStack` is statically allocated (not on the stack!), and that the value passed for the initial stack pointer is the location just beyond the end of the array. The first push will fill the last element of the array,

Model
Small
Medium
Large

1 Actually, the IN_DOS flag can be checked to see if a DOS call would be safe. This is an undocumented feature of DOS. A complete discussion of this subject is beyond the scope of this document.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Copyright © 1990 Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the written permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

Printed in the U.S.A. Specification and price change privileges are reserved.

Tektronix, Inc.
VXIbus Products
Jack Murdock Park
P.O. Box 3500
Vancouver, WA 98607-3500

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table of Contents

Section 1 General Information 1-1

Introduction 1-1

Notational Conventions 1-3

Section 2 TekVICS Programming Interface 2-1

Calling TekVICS From C 2-1

Error Codes 2-2

Section 3 TekVICS Message Delivery System 3-1

Device Naming and Addressing 3-2

Program Interface 3-3

Alphabetical Command Listing 3-3

Section 4 TekVICS BusManager 4-1

BusManager Context 4-1

Driver Considerations 4-2

Command Functions 4-3

Interrupt and Error Handling Functions 4-4

Commander Message Passing Functions 4-5

Event/Response Functions 4-5

Servant Message Passing Functions 4-6

Version Information Functions 4-6

Alphabetical Command Listing 4-7

Section 5 TekVICS Online Resource Manager 5-1

Introduction 5-1

Calling the OLRM From C 5-2

Alphabetical Command Listing 5-3

Section 6 TekVICS Advanced Trigger Functions 6-1

Introduction 6-1

ATF Functions 6-2

Alphabetical Function Listing 6-3

Advanced Trigger Functions Examples 6-28

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Appendix A I/O Control Commands A-1
GPIBRDG Control Commands A-1
VXIbus Bridge Control Commands A-8
VXIbus Word Serial Protocol A-8
Shared-Memory Protocol A-9

Appendix B Writing BusManager Applications B-1
Introduction B-1
Using the BusManager Interface B-1
The Bus Window B-2
Interrupts B-4
Polling Interrupts B-4
Interrupt Handlers B-6

Appendix C BusManager QuickC Applications C-1
Using the Microsoft® QuickC TekVICS Interface C-1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

List of Illustrations

Figure 1-1. VX5530/VX5535 software overview. 1-2

List of Tables

Table 2-1 Error Codes 2-2
TekVICS BusManager Functional Listing 4-3
Interrupts 4-4
Errors 4-4
ATF Functional Listing 6-2
Table A-1 GPIBRDG Control Commands A-2
Table A-2 Status Word Layout A-6
Table A-3 GPIB Error Codes A-7
Table A-4 GPIB Timeout Codes A-7
Table A-5 VXIbus Word Serial Control Commands A-8
Table A-6 VXIbus Shared-Memory Protocol Commands A-9

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Section 1
General Information

Introduction

This manual is written for C-language programmers using the TekVICS programming interface on a VX5530/VX5535. The reader is expected to have read the VX5530/VX5535 Users Manual for a global understanding of what is in TekVICS, how it is installed, and how to use BusProbe, a program that is very helpful in learning about the system. The reader is expected to have a basic knowledge of DOS.

The TekVICS programming environment provides programmers access to the TekVICS Message Manager, On Line Resource Manager (OLRM), and Advanced Trigger Functions (ATF).

The TekVICS Message Manager is a software package which coordinates all communication to instruments in the system. Currently, this includes both VXIbus and GPIB instruments. There are five parts to the Message Manager:

1) **Programming Interface Libraries.** The programming interface libraries are for the C and BASIC programming languages. In the C version, there are five different libraries. These are libraries for the small, medium, compact and large memory models for Microsoft® C, as well as a medium memory model library for Microsoft® Windows applications.

   The programming interface libraries provide access to the Message Delivery System and the BusManager.

2) **Message Delivery System (MDS).** This is the transport mechanism for application programs to communicate with intelligent instruments. Most programmers use the MDS facilities only.

3) **BusManager** . This is a lower-level facility for communicating over the VXIbus. It manages a bus window by which programs can directly read and write locations on the VXIbus. It also manages VXIbus interrupts and signals. Programmers should use the BusManager only for communicating with VXIbus register-based devices or VMEbus devices.

4) **GPIB Device Driver.** The GPIB device driver is a low-level driver controlling the GPIB hardware on the VX5530/VX5535. Programmers should never have to directly access this driver.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5) **Communication Bridges**. Bridges interface the Message Delivery System to the various interface busses and communication protocols supported by the VX5530/VX5535. There are three bridges. These are the *VXIbus Bridge* for communicating with VXIbus message-based devices using Word Serial or Shared Memory protocol, the *GPIB Bridge* for communicating with GPIB devices, and the *Command Bridge*. The *Command Bridge* represents local capabilities of the VX5530/VX5535 and does not communicate with any physical instruments. All bridges are implemented as terminate and stay resident (TSR) programs.

The Online Resource Manager (OLRM) provides the capability to manage VXIbus system resources during normal operation.

The Advanced Trigger Functions (ATF) provide advanced trigger, signal and timing features within a VXIbus mainframe, as well as the necessary real time links between multiple VXIbus subsystems.

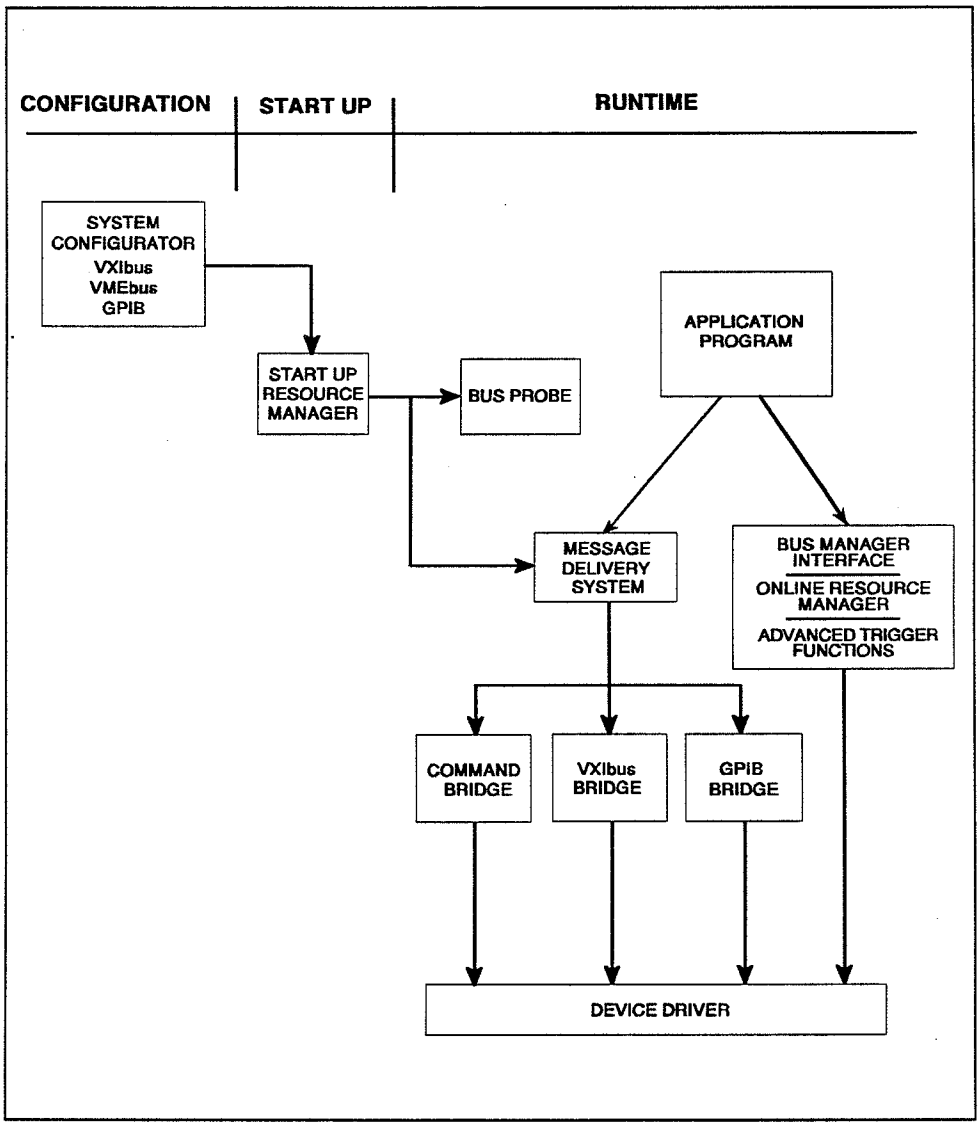


Figure 1-1. VX5530/VX5535 software overview.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Notational Conventions

The following notational conventions are used throughout this manual.

Convention	Description
Filename	The typeface shown at left is used in the text to refer to a DOS file name
`Example`	This typeface is used to simulate the output as seen on the VX5530/VX5535 screen. It is used for examples of commands, the output from commands, and code.
`[optional]`	The square brackets indicate optional fields.
<placeholder>	Words in angle brackets are placeholders for fields that you need to supply.
`(parameter)`	In the assembly language model discussion, words shown in parentheses in this typeface indicate the C language parameter for that assembly function.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Section 2
TekVICS Programming Interface

Each program-level function in TekVICS has a number of interfaces. Most have at least one interface for each of the supported applications languages, Microsoft® C and Microsoft® BASIC. For ease of programming, many functions have multiple interfaces.

The C language has its own interface library and corresponding definition files:

Library	Definition file
\tekvics\lib\vcmscl.lib	\tekvics\include\busmgr.h
\tekvics\lib\vcmscc.lib	\tekvics\include\vc_err.h
\tekvics\lib\vcmscm.lib	\tekvics\include\mds.h
\tekvics\lib\vcmscs.lib	\tekvics\include\olrm.h
\tekvics\lib\vcmscw.lib	\tekvics\include\gpibrdg.h
	\tekvics\include\vxicmd.h
	\tekvics\include\bdgioctl.h
	\tekvics\include\atf.h

The use of each of these files is discussed more fully below.

Calling TekVICS From C

The C language interface is designed to work with Microsoft® C compiler and libraries Version 5.0 or later. The C interface consists of the library files and header files. All Bus Interface Manager and Online Resource Manager calls are accessible through this interface.

Libraries **vcmscl.lib, vcmscc.lib, vcmscm.lib,** and **vcmscs.lib** are created for large, compact, medium, and small memory models. The **vcmscw.lib** library is a medium model library for Microsoft® Windows applications. The **busmgr.h** include file provides strong type checking and also does the conversion of near data to far data for programs using small (near code and near data), compact (near code and far data), or medium (far code and near data) models. The **vc_err.h** file contains all the error codes returned by the programming functions. **mds.h** contains the MDS function prototype and some constant definitions. On Line Resource Manager constant definitions and function prototype are in the **olrm.h** file. The definition files **gpibrdg.h** and **vxicmd.h** contain I/O control commands and define files for the GPIB and VXIbus bridges, respectively. I/O commands common to both the GPIB and VXIbus bridges are found in the **bdgioctl.h** file. The **atf.h** include file contains function prototypes and constant definitions for Advanced Trigger Functions.

The C libraries and include files can also be used to compile and link with Microsoft® QuickC programs. See Appendix B, *Writing BusManager Applications*, for more information.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Error Codes

Functions return a success (VC_SUCCESS) or failure indication. This status is typically returned as the value of the function. A negative value indicates an error, as shown in Table 2-1. These error codes are defined in the definition file, **vc_err.h.**

Table 2-1 Error Codes

Number	Error	Description
-1	ERR_FAIL	Undesignated error.
-2	ERR_TIMEOUT	A device failed to respond to a command (e.g., Byte Available command while sending data) within a period of time.
-3	ERR_BERR	A VXIbus error occurred while trying to access a device.
-4	ERR_WS	A word serial protocol error has been detected.
-5	ERR_RTIMEOUT	A device failed to generate a reply within a period of time.
-6	ERR_RBERR	After successfully sending data to a device, a VXIbus error occurred while trying to access a device to get a reply.
-20	ERR_BUFFER_FULL	The read buffer is insufficient to hold the incoming data.
-21	ERR_DUPLICATE_NAME	Attempted to add a duplicate name to the router table.
-23	ERR_INVALID_MESSAGE_TYPE	Call was made with an illegal message type.
-24	ERR_MEDIA_TABLE_FULL	No room exists in the media table to store the entry.
-25	ERR_ROUTER_TABLE_FULL	No room exists in the router table to store the entry.
-26	ERR_ILLEGAL_NAME	Call was made with an unrecognized device name. (The name either has invalid syntax or it is not in the routing table.)
-27	ERR_UNSUPPORTED_MEDIA	Call was made with an unknown media type.
-28	ERR_ILLEGAL_OPERATION	MDS was called with an illegal operation code.
-29	ERR_ILLEGAL_COMMAND	An illegal command was received by the command bridge.
-100	ERR_PARA	Parameter error.
-101	ERR_FOPEN	File cannot be opened for access.
-102	ERR_FCLOSE	Error when closing the file.
-103	ERR_FERROR	File read/write error.
-105	ERR_MEMORY	Memory allocation error.
-200	ERR_HW	Advanced Trigger Functions—hardware not available.
-201	ERR_RSNA	Advanced Trigger Functions—routing resource not available.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Section 3
TekVICS Message Delivery System

The Message Delivery System handles the delivery of commands, data write, and data read requests from a client on the VX5530/VX5535 to various server devices. It also handles the delivery of events/responses from server devices to a client on the VX5530/VX5535. A client may be:

- An application running in the VX5530/VX5535 (the most usual case)
- A GPIB gateway, translating requests from GPIB to VXIbus
- A network gateway, unwrapping requests from the network to VXIbus

The initial release of MDS supports only application programs as clients.

A server device may be:

- A VXIbus instrument
- A GPIB instrument
- An embedded servant, which is a local driver acting as a VXIbus device
- A network driver, wrapping VXIbus requests and sending them out the network

VXIbus and GPIB instrument support and provisions for writing embedded servants are provided with the initial release of MDS.

There are four types of information transferred

- Data Writes. These are bounded-size chunks of information that are transmitted from the client to the server. These operations are analogous to a write to a file.
- Data Reads. These are bounded-size chunks of information that are transmitted from the server to the client at the request of the client. The message delivery system does no interpretation of any of the transmitted messages.
- Commands. These are I/O control commands or communication protocol commands (e.g., word serial command in VXIbus or setup EOS in GPIB).
- Event/Response. These are 16-bit quantities that are transmitted asynchronously from the server to the client. An event/response is analogous to an interrupt. **This is the only information flow that is initiated by the server.**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Device Naming and Addressing

Devices are named by the Configurator and Start Up Resource Manager and comply with the device names defined in Section 7.6 of the IEEE 488.1 specification. As defined, device names are ASCII strings where each character in the string is in the set {0x41-0x5a ('A'-'Z'), 0x61-0x7a ('a'-'z'), 0x30-0x39 ('0'-'9'), 0x5f ('_')}. Names must start with a letter, and must not end with an underscore. The maximum device name length is 12 characters.

Messages are addressed to a device by concatenating a *device path* and the message together and sending it. A device path is defined as a list of device names separated by a colon. An optional colon may precede the first device name. This general form of addressing devices is known as *embedded addressing.*

For example, to send message "CLEAR" to device "dvm1" the appropriate line of C code would be:

```
ok = mpwrtmsg("dvm1","CLEAR\n", 0)
```

Note the newline on the end of the string! If the device is adhering to the IEEE 488.2 protocols then it requires a newline as the message terminator. The third parameter to `mpwrtmsg` is the message length. When passed a zero length, it counts the characters up to the first NULL (ASCII 0) and uses that as the message length. Since the "end of message" condition is sent with the last character, zero-length messages are not allowed.

The MDS looks only at the first name of the device path to route a message. This name is stripped off before the message is sent. If the destination device of the message is also a VXIbus commander with servants, the rest of the message may contain a further device path. For example, sending the message ":dvm1:scan: CLEAR" would cause the message ":scan:CLEAR" to be sent to device "dvm1", which would in turn send the message "CLEAR" to device "scan".

Just as messages are sent to a device by name, data from that device is read by referencing that device's name. For instance, to read data from "arb" in C:

```
mprdmsg("arb", buf, BUFLEN, &len)
```

where `buf` is the buffer into which the data is read, `BUFLEN` is the length of that buffer, and the size of the received message is returned in `len`.

When the MDS determines which device is being addressed, it calls the appropriate *bridge* for that device. A bridge can be thought of as a sort of device driver—it makes sure that the information gets output to (or input from) the specified device.

One important bridge is the *command bridge*. The command bridge is called when there is no address specified in the message. Its purpose is to handle commander-wide issues, such as returning the names of all the devices and their attributes.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Program Interface

Access to the MDS by application programs is through a collection of library routines. These routines allow applications to establish routes and communicate with the various devices in the system. The library routines are defined with **far pascal** calling conventions to simplify application development. All error values returned by the library routines are defined in the **vc_err.h** header file.

The program **sample.c**, in the **\tekvics\sample** directory, illustrates how to use the MDS functions.

Alphabetical Command Listing

Commands listed below with a brief summary statement are described in detail, in alphabetical order, on the following pages.

mpcheck	Checks whether MDS is loaded
mpctrl	I/O control call for the device
mpdtof	Reads data from device and writes to file
mpevent	Sets Event/Response handler for a device
mpftod	Reads data from file and writes to device
mppollevent	Polls event from the event queue
mprdmsg	Receives message from device
mpsendsig	Sends a signal/event to device
mpwrtmsg	Sends message to device

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

mpcheck

mpcheck

Purpose: Checks whether MDS is loaded.

Syntax:

```
short far pascal mpcheck();
```

Description: This function determines whether or not MDS is loaded.

Return Value: Returns `VC_SUCCESS` if MDS is loaded.

Errors: Returns `ERR_FAIL` if MDS is not loaded.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

mpctrl

mpctrl

Purpose: Performs an I/O control (ioctl) call for the device.

Syntax:

```
short far pascal mpctrl(devicename, request, argp, result);
unsigned char far *devicename; /* device name */
unsigned long request;         /* type of ioctl call */
unsigned void far *argp;       /* argument of the ioctl call */
unsigned long far *result;     /* result of the ioctl call */
```

Description: Performs an I/O control call for the destination device. The destination device is specified by `devicename`. For each type of control specified by the `request` field, ioctl arguments are provided by `argp`. The `argp` may be a structure type which may differ for different requests. See Appendix A for more information on I/O control calls.

Return Value: Returns `VC_SUCCESS` if successful. Otherwise a negative value is returned, as noted below.

Errors:

`ERR_ILLEGAL_NAME`	invalid device name
`ERR_FAIL`	other error
`ERR_TIMEOUT`	timeout
`ERR_BERR`	bus error
`ERR_WS`	word serial protocol error
`ERR_RTIMEOUT`	timeout on receive
`ERR_RBERR`	bus error on receive

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

mpdtof mpdtof

Purpose: Reads data from device and writes to file.

Syntax:

```
short far pascal mpdtof(devicename, filename, maxsize,
                        rsize, access);
unsigned char far *devicename; /* device name */
unsigned char far *filename;   /* file to be written to */
long maxsize;                  /* maximum number of bytes
                                  to read */
unsigned long far *rsize;      /* number of bytes read */
short access;                  /* file access code */
```

Description: Reads data from device and writes to a file. If the file does not exist, it will be created. If the file exists, it will be overwritten if the access code is F_DEFAULT. If the file exists and the access code F_APPEND is specified, then the data will be appended to the current file.

Return Value: Returns VC_SUCCESS if successful. Otherwise a negative value is returned, as noted below.

Errors:

ERR_ILLEGAL_NAME	invalid device name
ERR_TIMEOUT	timeout
ERR_BERR	bus error
ERR_WS	word serial protocol error
ERR_FOPEN	file cannot be opened for access
ERR_FCLOSE	error when closing the file
ERR_PARA	parameter error
ERR_RTIMEOUT	timeout on receive
ERR_RBERR	bus error on receive

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

mpevent

mpevent

Purpose: Sets Event/Response handler for a device.

Syntax:

```
short far pascal mpevent(devicename, erfunction,
                         olderfunction, devicecode, olddevicecode);
unsigned char far *devicename; /* device name */
void (far *erfunction)();      /* event/response handler */
void (far **olderfunction)();  /* previous event/response
                                  handler */
unsigned long devicecode;      /* device code */
unsigned long *olddevicecode;  /* previous device code */
```

Description: Sets the Event/Response handler for a device. Upon receiving an event from the device, the `erfunction` will be called and passed the `devicecode` and `eventID`.

```
(*erfunction)(devicecode, eventID)
```

The event ID is the 16-bit identification of the event or response. The Event/Response handler is assigned to the device specified by the device name. The device code is any number associated with a device (this allows a single Event/Response handler to handle different devices). The previous Event/Response handler and device code is returned so that the application can restore it.

Return Value: Returns `VC_SUCCESS` if successful. Otherwise a negative value is returned, as noted below.

Errors:

`ERR_ILLEGAL_NAME`	invalid device name
`ERR_FAIL`	other error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

mpftod

mpftod

Purpose: Reads data from file and writes to device.

Syntax:

```
short far pascal mpftod(devicename, filename);
unsigned char far *devicename; /* device name */
unsigned char far *filename;   /* file to be read from */
```

Description: Reads data from file and writes to the device. The file should be able to be opened for a read operation. The file is opened in binary mode.

Return Value: Returns VC_SUCCESS if successful. Otherwise a negative value is returned, as noted below.

Errors:

ERR_ILLEGAL_NAME	invalid device name
ERR_TIMEOUT	device timeout
ERR_FOPEN	file cannot be opened for access
ERR_BERR	bus error
ERR_WS	word serial protocol error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

mppollevent

mppollevent

Purpose: Polls event from the event queue.

Syntax:

```
short far pascal mppollevent(devicename, eventID);
unsigned char far *devicename; /* device name */
unsigned short far *eventID;   /* event ID */
```

Description: This function polls the event queue to determine whether an event has been received. The device name is the name assigned to a device.

Return Value: Returns a value greater than 0 if an event is dequeued from the event queue. If the event queue is empty or there is no event for the device, 0 is returned. Otherwise, a negative error value is returned, as noted below.

Errors:

ERR_ILLEGAL_NAME	invalid device name
ERR_FAIL	Undesignated error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

mprdmsg mprdmsg

Purpose: Receives message from device.

Syntax:

```
short far pascal mprdmsg(devicename, str, length, rsize);
unsigned char far *devicename; /* device name */
unsigned char far *str;        /* received message */
unsigned long length;          /* message buffer size */
unsigned long far *rsize;      /* number of bytes received */
```

Description: Receives message from the destination device. The destination device is specified by the device name. It will read the data until it receives the end of message character/signal (depending on communication protocol), or until the number of bytes received equals the `length` specified.

The user can check whether a complete message has been received by checking the return value of this function. If a multiple read is required to read a complete message, then the user has to manage the receive buffers. If `rsize` is non-NULL, then the number of bytes received will be returned. The minimum receiving buffer size is 1.

Return Value: Returns `VC_SUCCESS` if a complete message has been received. If the return value is less than 0, an error has occurred. If the return value is `ERR_BUFFER_FULL`, the receiving buffer is full and the end of message character/signal was not received.

Errors:

`ERR_ILLEGAL_NAME`	invalid device name
`ERR_BUFFER_FULL`	receiver buffer full
`ERR_TIMEOUT`	device timeout
`ERR_PARA`	parameter error
`ERR_BERR`	bus error
`ERR_WS`	word serial protocol error
`ERR_RTIMEOUT`	timeout on receive
`ERR_RBERR`	bus error on receive

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

mpsendsig

mpsendsig

Purpose: Sends a signal/event to a device.

Syntax:

```
short far pascal mpsendsig(devicename, eventID);
unsigned char far *devicename; /* device name */
unsigned short eventID; /* event ID */
```

Description: Sends a signal/event to a device. If the device is a VXIbus instrument, the event ID value is written to the signal register on the device. If the device is not a VXIbus instrument, an error is flagged.

Return Value: Returns `VC_SUCCESS` if successful. Otherwise a negative value is returned, as noted below.

Errors:

`ERR_ILLEGAL_NAME`	invalid device name
`ERR_FAIL`	other error
`ERR_BERR`	bus error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

mpwrtmsg

mpwrtmsg

Purpose: Sends message to device.

Syntax:

```
short far pascal mpwrtmsg(devicename, str, length);
unsigned char far *devicename; /* device name */
unsigned char far *str;        /* message to send */
unsigned long length;          /* length of message */
```

Description: Sends message to the destination device. The destination device is specified by the device name. The number of bytes to send is specified by the `length` parameter.

Messages with NULL names are routed to the Command Bridge. A subsequent call to `mprdmsg` with a NULL name retrieves the data.

Return Value: Returns `VC_SUCCESS` if successful. Otherwise a negative value is returned, as noted below.

Errors:

`ERR_ILLEGAL_NAME`	invalid device name
`ERR_TIMEOUT`	device timeout
`ERR_PARA`	parameter error
`ERR_BERR`	bus error
`ERR_WS`	word serial protocol error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Section 4
TekVICS BusManager

The BusManager is the part of the Message Manager that handles the proprietary VXIbus interface hardware.

It is intended that **all** bus activity through the bus window be coordinated by the BusManager rather than going to the underlying hardware, so that

- Software is easier to write
- Drivers for various devices on the bus won't interfere with one another
- Compatibility is maintained with different implementations of the bus interface hardware

The BusManager provides facilities for reading and writing locations on the bus, catching or causing interrupts on the bus, and transferring of data. It is accessible from either user or interrupt level. The MDS itself uses the BusManager.

NOTE

The BusManager interface should only be used for communication with VXIbus register-based devices and VMEbus devices.

BusManager Context

There are three parts to the context of the BusManager:

1) Vector of interrupt routines assigned to bus interrupts. This is fairly static.

2) Bus access **mode** which encompasses address space (A16, A24, A32), and byte sex. This is dynamic.

3) The **base** of the bus mapping window—at what bus addresses it is looking. This is dynamic.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The primary means of accessing a device is via its control and status registers. These registers are addressable locations, usually in VMEbus A16 space. They are accessed through the VMEbus window. The VMEbus window is a 64KB region of memory within the first megabyte of memory on the VX5530/VX5535 (and therefore easily accessible) which can be mapped to arbitrary sections of the A16, A24, or A32 address spaces on 64KB boundaries. The bus window is only a *master* to the VMEbus—it has no slave address itself and cannot be used as the destination of accesses originated on other boards. This means, for instance, that you can't have a VMEbus device DMA into the bus window.

The **mode** and **base** together form the **state** of the BusManager. This state is part of the context of the current process, just like the contents of the processor registers are part of the context of the current process. The BusManager handles the saving and restore of this state for VXIbus interrupts; however, if a VMEbus interrupt handler changes the BusManager state (*i.e.*, to access a device on the VXIbus), it must save the previous state before and restore it after. Similarly, in multitasking environments like Windows/386 or DESQview the program must save the state before potential context switches (*e.g.*, waiting for input) and restore it afterwards.

An exception to the rule about saving and restoring state is the case where only A16 space is being used and the bus access mode isn't changed. In this case the BusManager state is never changed since the VXIbus mapping window can encompass all of A16 space without changing the mapping.

Driver Considerations

To speed state changes in interrupt handlers that access the VXIbus, it is suggested that during initialization the driver set up the appropriate bus mapping and then call save state (`vc_save_state`) to make an internal format descriptor for the mapping. On entry to the interrupt handler, a call to restore state (`vc_rest_state`), passing that internal format descriptor will quickly set the mapping required by the driver. The previous state is stored before calling the handler and restored after calling the handler by the BusManager, so the interrupt handler doesn't have to worry about it.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Command Functions

Following is a functional listing of TekVICS BusManager commands, and a general discussion of selected functions. Detailed descriptions of all commands follow later in this section.

Functional Listing

VXIbus and Internal Control	VXIbus and Internal Information
vc_vme_ctrl( ) vc_vxi_ctrl( )	vc_bios_ver( ) vc_bm_ver( ) vc_ck_bm( ) vc_get_slavebase( ) vc_get_ula( ) vc_hw_ver( )
Event/Response Handling	**Error Handling**
vc_er_get( ) vc_er_que( ) vc_er_redir( ) vc_er_unredir( )	vc_dis_err( ) vc_en_err( ) vc_get_error( ) vc_set_error( ) vc_watchdog( )
VMEbus Data Transfer	**Interrupt Handling**
vc_from_vme( ) vc_set_acc_mode( ) vc_map_bus( ) vc_rest_state( ) vc_save_state( ) vc_get_acc_mode( ) vc_to_vme( )	vc_ck_intr( ) vc_dis_intr( ) vc_en_intr( ) vc_get_intr( ) vc_set_intr( ) vc_sig_intr( ) vc_wait_intr( ) vc_wait_intr2( )
Commander Word Serial Protocol	**Slave Word Serial Protocol**
vc_elws_cmd( ) vc_lws_cmd( ) vc_ws_cmd( ) vc_ws_rcv_str( ) vc_ws_snd_str( ) vc_ws_stat( )	vc_er_serv_intr( ) vc_er_serv_sig( ) vc_ws_serv_arm( ) vc_ws_serv_peek( ) vc_ws_serv_peek2( ) vc_ws_serv_rcv( ) vc_ws_serv_rcv2( ) vc_ws_serv_send( ) vc_ws_serv_send2( )

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Interrupt and Error Handling Functions

The term handler refers to a subroutine that is called when an interrupt or error occurs. This comparatively low-level passing of control requires that the handler itself be in memory and that it obey some rather strict rules, but it allows nearly instantaneous response to other devices.

Interrupts

Code	Name	Indicates
0	Message	VXIbus word serial servant response received
1	VXIbus interrupt 1	Interrupt asserted by VXIbus device
2	VXIbus interrupt 2	Interrupt asserted by VXIbus device
3	VXIbus interrupt 3	Interrupt asserted by VXIbus device
4	VXIbus interrupt 4	Interrupt asserted by VXIbus device
5	VXIbus interrupt 5	Interrupt asserted by VXIbus device
6	VXIbus interrupt 6	Interrupt asserted by VXIbus device
7	VXIbus interrupt 7	Interrupt asserted by VXIbus device
8*	Event/Response	Event or response has been received in the signal register or via redirected interrupts

*The Event/Response interrupt is used by MDS. Do not assign a handler to this interrupt unless you are replacing the MDS with another VXIbus communication mechanism.

Errors

Code	Name	Indicates
0	SYSFAIL	VXIbus failure signal
1	BERR	VXIbus transaction error signal. Usually indicates an access to non-existent memory
2	ACFAIL	VXIbus signal from the power supply indicating that shutdown is imminent
3	Watchdog	Watchdog timer has elapsed

Interrupt and error handlers are specified by a **far** function address (initial CS:IP) and a **far** pointer to the base of the stack used by the error handler (initial SS:SP). A handler is invoked by switching to the specified stack and then doing a far call to the function. C language parameter passing is used: parameters are pushed on the stack before the handler is called and popped after the handler returns.

A handler may call BusManager functions; however, handlers generally may not call DOS functions or use the C library function `longjmp()` to change the flow of control in the interrupted routine.

A handler may enable interrupts; however, interrupt handlers themselves are not reentrant (due to the specification of the stack base) .

For a detailed description of how to use interrupts, refer to Appendix B, *Writing BusManager Applications*.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Commander Message Passing Functions

The VXIbus message functions are:

- Word Serial Protocol
- Longword Serial Protocol
- Extended Longword Serial Protocol
- Error Handling for the Serial Protocols utilizing the ERR* bit in the Bus Response Register (BRE)
- Signals and Interrupts

Event/Response Functions

VXIbus *events* and *responses* (collectively called *ER's*) get special handling. They arrive through the signal register or as the Status/ID returned in response to an interrupt acknowledge for a VXIbus interrupt. All ER's are queued together, in order, as the sequence of responses and events may be important.

When a value is placed in the signal register, the signal FIFO is emptied into the BusManager-maintained ER queue. The BusManager uses the hardware signal interrupt internally to maintain this queue. Interrupts may be designated to be VXIbus interrupts so that the Status/ID's returned in response to interrupt acknowledges are recognized as ER's and placed in the ER queue as well.

The *ER Interrupt* is asserted by the BusManager when the ER queue is non-empty. A handler can be assigned to the ER Interrupt using the `vc_set_intr()` function. This interrupt is enabled and disabled via the `vc_en_intr()` and `vc_dis_intr()` functions. The `vc_wait_intr()` function can be used to poll the ER Interrupt.

Programs may call a function that dequeues and returns the oldest message in the ER queue.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Servant Message Passing Functions

The BusManager provides the lowest-level interface for messaging operations. These functions support the VX5530/VX5535 as a VXIbus servant.

These functions include facilities for sending Event/Responses to the VX5530/VX5535 commander via VXIbus interrupts or the commander's signal register, and operations for reading/writing to/from the VX5530/VX5535 message registers.

Messages are 16- or 32-bit values that can be *sent* (written in the message registers to be read by another master on the VMEbus) or *received* (read from the message registers after previously having been written by another VMEbus master. For the purposes of this document, we refer to a local send of a message as *putting* a message (since the message is "put" into the local registers).

The VX5530/VX5535 bus interface provides support for message passing in the form of location monitors and a message interrupt. Note that the message interrupt is defined such that the message interrupt is asserted when nothing is happening (neither waiting to send nor receive messages).

Version Information Functions

Version information is available for the BusManager, VX5530/VX5535 hardware, and BIOS. A function that indicates whether the BusManager is currently loaded in the system is also provided.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Alphabetical Command Listing

Commands listed below with a brief purpose statement are described in detail, in alphabetical order, on the following pages.

vc_bios_ver	Returns version number for BIOS.
vc_bm_ver	Returns version number for BusManager.
vc_ck_bm	Checks if BusManager is loaded.
vc_ck_intr	Reads the number of the interrupt currently being asserted by the VX5530/VX5535.
vc_dis_err	Disables the specified error without affecting the handler assignment.
vc_dis_intr	Disables the specified interrupt without affecting the handler assignment.
vc_elws_cmd	Sends one extended long word serial command.
vc_en_err	Enables the specified error without affecting the handler assignment.
vc_en_intr	Enables the specified interrupt without affecting the handler assignment.
vc_er_get	Gets the oldest event/response from the ER queue.
vc_er_que	Puts the supplied value as the newest element in the ER queue.
vc_er_redir	Assigns specified interrupt to be the VXIbus event/response interrupt.
vc_er_serv_intr	Sends an ER from the VX5530/VX5535 as a servant to a commander, using a VXIbus interrupt.
vc_er_serv_sig	Sends signal to VX5530/VX5535's commander.
vc_er_unredir	Deassigns the specified interrupt as a VXIbus event/response interrupt.
vc_from_vme	Copies data from VMEbus space to the VX5530/VX5535 used as a controller.
vc_get_acc_mode	Gets current VMEbus access mode.
vc_get_error	Returns entry and stack base addresses of current error handler.
vc_get_intr	Gets address of interrupt handler.
vc_get_slavebase	Returns the current slave memory base address.
vc_get_ula	Reports VX5530/VX5535 controller's logical address.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_hw_ver	Returns version number for hardware.
vc_lws_cmd	Sends long word serial command.
vc_map_bus	Maps VMEbus address to internal controller address.
vc_rest_state	Restores the previously-saved VMEbus access state.
vc_save_state	Saves access mode and bus window base as the VMEbus access state.
vc_set_acc_mode	Sets VMEbus access mode.
vc_set_error	Defines error handlers.
vc_set_intr	Connects a bus interrupt to an interrupt routine on the VX5530/VX5535.
vc_sig_intr	Signals a bus interrupt.
vc_to_vme	Copies data from VX5530/VX5535 controller memory to VMEbus space.
vc_wait_intr **vc_wait_intr2**	Waits for interrupt before executing caller. The **2** version differs slightly (see command description).
vc_watchdog	Sets specified watchdog timer service.
vc_ws_cmd	Sends one word serial command.
vc_ws_rcv_str	Receives word serial string via BYTE REQUEST.
vc_ws_serv_arm	Arms local VXIbus data register for command receive function.
vc_ws_serv_peek **vc_ws_serv_peek2**	Peek command. Same as **vc_ws_serv_rcv** but does not remove incoming command. The **2** version differs slightly (see command description).
vc_ws_serv_rcv **vc_ws_serv_rcv2**	The VX5530/VX5535 as a servant waits for a command to arrive and returns it. The **2** version differs slightly (see command description).
vc_ws_serv_send **vc_ws_serv_send2**	Sends a word serial command from this servant to its commander. The **2** version differs slightly (see command description).
vc_ws_snd_str	Sends word serial string via BYTE AVAILABLE.
vc_ws_stat	Gets designated servant's word serial status.
vc_vme_ctrl	VMEbus control. Allows reading, setting and clearing of various control bits in the VMEbus subset of the VXIbus interface.
vc_vxi_ctrl	VXIbus control. Allows reading, setting and clearing of various control bits in the VXIbus interface.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_bios_ver

vc_bios_ver

Purpose: This function returns the version number for the BIOS.

Syntax:

```
short far pascal vc_bios_ver();
```

Description: The BIOS version number consists of the major and minor version numbers of the BIOS that is installed in the VX5530/VX5535.

Returns: Appropriate version number.

Errors: None.

Assembly: *Entry:*

AH =BM_VER

Returns:

CH Major BIOS Version Number
CL Minor BIOS Version Number

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_bm_ver

vc_bm_ver

Purpose: This function returns the version number for the BusManager.

Syntax:

```
short far pascal vc_bm_ver();
```

Description: The BusManager version number consists of a major version and minor version number assigned to the BusManager software currently running in the VX5530/VX5535.

Returns: Appropriate version number.

Errors: None.

Assembly: *Entry:*

AH =BM_VER

Returns:

AH Major BusManager Version Number
AL Minor BusManager Version Number

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_ck_bm

vc_ck_bm

Purpose: Check BusManager.

Syntax: `short far pascal vc_ck_bm();`

Return Value: Returns `VC_SUCCESS` if the BusManager is loaded.

Errors: Returns `ERR_FAIL` if the BusManager is not loaded.

Assembly: None.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_ck_intr vc_ck_intr

Purpose: Checks VXIbus interrupt.

Syntax:
```
short far pascal vc_ck_intr();
```

Description: Reads the number of the interrupt currently being asserted by this VX5530/VX5535.

Remember that interrupt acknowledgement is a hardware handshake and does not indicate that the remote interrupt handling code has executed.

Return Value: The number of the interrupt currently asserted. If no interrupt is being asserted (*e.g.,* the last interrupt has been acknowledged) then zero is returned.

Errors: None

Assembly: *Entry:*

AH = BM_CKBI

Returns:

AL VXIbus interrupt (1..7) *or* 0 for no interrupt being asserted

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_dis_err vc_dis_err

Purpose: Disables the specified error without affecting the handler assignment.

Syntax:

```
short far pascal vc_dis_err(err_code);
short err_code;   /* bus event code */
```

Description: The associated handler will not be called if an error condition occurs.

Use one of the following error numbers, as defined in **busmgr.h**, for `err_code`.

```
/*
 *  Error Constants
 */

# define BM_SYSFAIL_ERR                (0)
# define BM_BERR_ERR                   (1)
# define BM_ACFAIL_ERR                 (2)
# define BM_WATCHDOG_ERR               (3)
```

Return Value: Returns `VC_SUCCESS` if successful.

Errors: Returns `ERR_FAIL` if error occurs.

Assembly: *Entry:*

AH	=	BM_DERR
AL	Bus event code (`err_code`)	
	0	SysFail
	1	Bus error (BERR)
	2	ACFAIL
	3	Watchdog timer

Returns:

Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_dis_intr vc_dis_intr

Purpose: Disables the specified interrupt without affecting the handler assignment.

Syntax:

```
short far pascal vc_dis_intr(interrupt);
short interrupt;  /* VXIbus interrupt */
```

Description: This function disables the specified interrupt without affecting the handler assignment. The parameter `interrupt` is (1..7) for VXIbus interrupt, 0 for message interrupt, or 8 for Event/Response pending.

Return Value: Returns `VC_SUCCESS` if successful.

Errors: Returns `ERR_FAIL` if error occurs.

Assembly: *Entry:*

AH	=	BM_DINT
AL		VXIbus interrupt (1..7), 0 for message interrupt, or 8 for Event/response pending `(interrupt)`

Returns:

Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_elws_cmd vc_elws_cmd

Purpose: Sends one extended longword serial command.

Syntax:

```
short far pascal vc_elws_cmd(ula, command, ticks);
unsigned short ula;                    /* servant's unique logical
                                          addr */
unsigned short far command[3]; /* command to send */
unsigned short ticks;                  /* timeout in 18.2 Hz ticks */
```

Description: A command will be sent only when the servant device's WRDY bit is set.

NOTE

Extended longword serial commands do not generate a reply.

Return Value: Returns `VC_SUCCESS` if successful or a negative number on failure.

Errors: Returns one of the following:

`ERR_TIMEOUT`	TimeOut
`ERR_BERR`	Bus Error
`ERR_WS`	Word Serial Protocol Error

Assembly: *Entry:*

AH	= BM_ELWS_CMD
AL	Servant's ULA (`ula`)
ES:SI	Address of command to send (`command`)
CX	Number of 18.2Hz clock ticks to send the command (`ticks`)

Returns:

AL	Return code. One of
	`VC_SUCCESS`
	`ERR_TIMEOUT`
	`ERR_BERR`
	`ERR_WS`

CX	Number of clock ticks remaining
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_en_err

vc_en_err

Purpose: Enables the specified VXIbus error without affecting the error handler assignment.

Syntax:

```
short far pascal vc_en_err(err_code);
short err_code;   /* bus event code */
```

Description: This should be used only in conjunction with the `vc_dis_err` command . No check is made to make sure there is a handler assigned to the error.

Use one of the following error numbers, as defined in **busmgr.h**, for `err_code`.

```
/*
 *  Error Constants
 */

# define BM_SYSFAIL_ERR                    (0)
# define BM_BERR_ERR                       (1)
# define BM_ACFAIL_ERR                     (2)
# define BM_WATCHDOG_ERR                   (3)
```

Return Value: Returns `VC_SUCCESS` if successful.

Errors: Returns `ERR_FAIL` if error occurs.

Assembly: *Entry:*

AH	=	BM_EERR
AL	Bus event code (`err_code`)	
	0	SysFail
	1	Bus error (BERR)
	2	ACFAIL
	3	Watchdog timer

Returns:

Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_en_intr vc_en_intr

Purpose: Enables the specified interrupt without affecting the handler assignment.

Syntax:

```
short far pascal vc_en_intr(interrupt);
short interrupt;  /* VXIbus interrupt */
```

Description: This should be used only in conjunction with `vc_dis_intr`. No check is made to make sure there is a handler assigned to the interrupt. The parameter `interrupt` is (1..7) for VXIbus interrupt, 0 for message interrupt, or 8 for Event/Response pending.

Return Value: Returns `VC_SUCCESS` if successful.

Errors: Returns `ERR_FAIL` if error occurs.

Assembly: *Entry:*

AH = BM_EINT
AL VXIbus interrupt (1..7), 0 for message interrupt, or 8 for Event/response pending `(interrupt)`

Returns:

Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_er_get vc_er_get

Purpose: This function dequeues and returns the oldest ER.

Syntax:
```
short far pascal vc_er_get(er_ptr);
unsigned short far *er_ptr; /* addr of where to return ER */
```

Description: The oldest ER will be put into `er_ptr`. If the returned value is the last ER in the queue, the ER interrupt is deasserted.

NOTE

This function is used by the MDS. It is recommended that you do not call this function unless you are replacing MDS with another communications mechanism.

Return Value: Returns `TRUE` if the queue is non-empty and `FALSE` (0) if the queue is empty..

Errors: None.

Assembly: *Entry:*

AH	=	BM_ER_GET
ES:DI	Pointer to two-byte area in which to place dequeued ER (`er_ptr`)	

Returns:

AL	Return code. One of	
	TRUE (non-zero)	If ERreturned
	FALSE (zero)	If queue was empty
Carry Flag	Set if error	

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_er_que vc_er_que

Purpose: Enqueues the supplied value as the newest element in the ER queue.

Syntax:
```
short far pascal vc_er_que(er);
unsigned short er;      /* ER value to be queued */
```

Description: When the value is queued, the ER interrupt is asserted (since the ER queue is now non-empty) and, if enabled, the ER interrupt handler will be called before this function returns.

Return Value: Returns `TRUE` if value was queued, and `FALSE` (0) if the queue is full.

Errors: None.

Assembly: *Entry:*

AH	=	BM_ER_QUE
BX		Value to be queued `(er)`

Returns:

AL	Return code. One of	
	`TRUE` (non-zero)	If value was queued
	`FALSE` (zero)	If queue was full so value could not be queued
Carry Flag	Set if error	

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_er_redir vc_er_redir

Purpose: Designates an interrupt to be a VXIbus Event/Response interrupt for receipt of events and responses from servants.

Syntax:

```
short far pascal vc_er_redir(interrupt);
short interrupt;  /* VXIbus interrupt from which to redirect
                     ER's */
```

Description: When an interrupt is redirected, the interrupt is unmasked.

At system restart no interrupts are redirected. Any number (1-7) of interrupts may be redirected.

An interrupt may not be both redirected and have a handler assigned to it.

After a redirected interrupt is asserted and acknowledged, the low 16-bits of the returned Status/ID is placed in the ER queue. An ER interrupt is then asserted (as the queue is now non-empty) .

Return Value: Returns `VC_SUCCESS` if successful.

Errors: `ERR_FAIL` will be returned if the interrupt selected to be redirected is currently assigned a handler.

Assembly: *Entry:*

AH = BM_ER_REDIR
AL Interrupt number (1..7) `(interrupt)`

Returns:

AL Return code. One of
`VC_SUCCESS`
`ERR_FAIL`
Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_er_serv_intr vc_er_serv_intr

Purpose: Sends an ER to a commander utilizing a VXIbus interrupt.

Syntax:

```
short far pascal vc_er_serv_intr(interrupt, er);
short interrupt;          /* VXIbus interrupt to send this
                             ER thru */
unsigned short er;        /* ER value to send */
```

Description: This function is for implementing a VXIbus servant interface on the VX5530/VX5535.

In the VX5530/VX5535, the event/response value sent via an interrupt must be stored in the message register that the VX5530/VX5535 uses for word serial communication with its commander before the interrupt is signaled. If there is no word serial activity pending, then the event/response value is just stored in the register and the operation proceeds. If a word serial command from the commander is in the register, then the command is saved and the register used. If there is outgoing data in the register, then this function must wait until the commander has read the data before signaling the interrupt. These various contortions are transparent to the caller.

Return Value: Returns `VC_SUCCESS` if successful.

Errors: `ERR_FAIL` indicates the VX5530/VX5535 is unable to send because an interrupt is currently being asserted. `ERR_TIMEOUT` indicates a time out when sending the ER.

Assembly: *Entry:*

AH	=	BM_ER_SNDINTR
AL		Interrupt by which to send Event/Response `(interrupt)`
BX		Event/Response value to send `(er)`

Returns:

AL	Return code. One of VC_SUCCESS ERR_TIMEOUT ERR_FAIL
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_er_serv_sig vc_er_serv_sig

Purpose: Signals the VX5530/VX5535's commander by placing a value in its signal register.

Syntax:

```
short far pascal vc_er_serv_sig(ula, er);
unsigned short ula;      /* unique logical addr of commander */
unsigned short er;       /* ER value to send */
```

Description: This function is for implementing a VXIbus servant interface on the VX5530/VX5535.

Return Value: Returns `VC_SUCCESS` if successful.

Errors: `ERR_BERR` or `ERR_FAIL` if the commander has no signal register.

Assembly: *Entry:*

AH = BM_ER_SNDSIG
AL Unique logical address of the Commander to which to send the signal `(ula)`
BX Event/Response value to send `(er)`

Returns:

AL Return code. One of
`VC_SUCCESS`
`ERR_BERR`
`ERR_FAIL`
Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_er_unredir vc_er_unredir

Purpose: De-assigns an interrupt as a VXIbus Event/Response interrupt and makes it available as a regular bus interrupt.

Syntax:

```
short far pascal vc_er_unredir(interrupt);
short interrupt;  /* VXIbus interrupt number previously
                     redirected */
```

Description: None.

Return Value: Returns `VC_SUCCESS` if successful.

Errors: `ERR_FAIL` will be returned if there is an attempt to un-redirect an interrupt that is not redirected.

Assembly: *Entry:*

AH = BM_ER_UNREDIR
AL Interrupt number (1..7) `(interrupt)`

Returns:

AL Return code. One of
`VC_SUCCESS`
`ERR_FAIL`
Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_from_vme

vc_from_vme

Purpose: Copies data from VMEbus space to VX5530/VX5535 controller.

Syntax:

```
unsigned short far pascal vc_from_vme(width, source,
                          dest, count);
short width;              /* copy width */
unsigned long source;     /* VXIbus address */
char far *dest;           /* starting destination address
                          in VX5530 memory */
unsigned short count;     /* byte count to transfer */
```

Description: Efficiently copies a block of memory to VX5530/VX5535 memory from consecutive bus locations, using the current access mode. This is intended for communicating between two VX5530/VX5535s, or dealing with devices such as network cards that map their memory onto the bus.

The `(width)` parameter specifies how the data is moved: whether data is to be moved in 8-, 16-, or 32-bit blocks, and whether error checking is performed (see below). VXIbus transfers are always done on natural boundary alignment—16-bit quantities are read from the VXIbus only at even addresses, and 32-bit quantities are read from the VXIbus only at addresses divisible by 4. Transfers from non-aligned locations are done by reading a block and extracting the appropriate data. For instance, with a 32-bit copy from a non-aligned address, a leading 32-bit word would be read and the leading bytes copied from it, then whole 32-bit values would be transferred, finally a trailing word would be read and trailing bytes extracted.

If the Fast Copy (C `define BM_FASTCOPY`) is not set in `(width)`, BERR is checked after every transfer. If there is an error, the copy is aborted and the BERR error handler is not called. Any errors are reflected by a non-zero return value of this function. This eliminates the need of requiring the program calling this function to coordinate with the BERR handler.

If the Fast Copy bit *is* set in `(width)`, only transfers to non-aligned locations (if any) are checked. This option takes advantage of the pipelining in the processor and the VXIbus interface hardware to achieve a five times improvement in transfer rate; however, errors may go undetected.

Note that `(count)` always contains the size of the transfer *in bytes* regardless of the width of the transfer. If `(count)` = 0, nothing will be transferred.

Any VXIbus mapping in place when this function is called is restored on return.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The following manifest constants are defined in **busmgr.h** to make life easier:

```
/*
*   Copy defines
*/
# define BM_W8          (1)         /* 8-bit xfer width */
# define BM_W16         (2)         /* 16-bit xfer width */
# define BM_W32         (4)         /* 32-bit xfer width */
# define BM_FASTCOPY    (0x80)      /* fast copy bit */
```

Return Value: Returns `VC_SUCCESS` (zero) if the copy is successful, otherwise it's the number of bytes remaining when a bus error occurs.

Errors: Any errors are reflected by a non-zero return value.

Assembly: *Entry:*

AH	= BM_CPIN
AL	Copy width `(width)`
	1 8-bit wide copy
	2 16-bit wide copy
	4 32-bit wide copy
	Optionally ORed with
	80H Fast Copy bit
ES:DI	Starting destination address in VX5530/VX5535 memory `(dest)`
BX	High 16-bits of VXIbus source address `(source)`
SI	Low 16-bits of VXIbus source address `(source)`
CX	Byte count to transfer, maximum of 64K-1 bytes `(count)`

Returns:

CX	= `VC_SUCCESS` if copy successful, otherwise number of bytes remaining on bus error
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_get_acc_mode vc_get_acc_mode

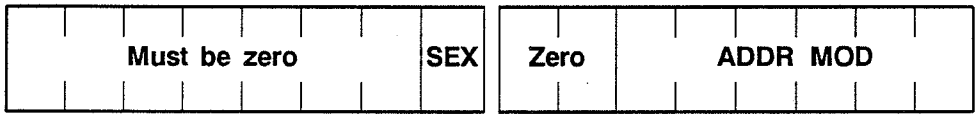
Purpose: This gets the bus access parameters currently in effect.

Syntax:

```
short far pascal vc_get_acc_mode();
```

Description: This command uses the access mode constants defined in **busmgr.h**. See `vc_set_acc_mode`.

Return Value: Current mode bits, in the format shown below.



Where:

SEX	= 0 for Little Endian (386-style) = 1 for Big Endian (68000-style)
ADDR MOD	VXIbus address modifier

Errors: None.

Assembly: *Entry:*

AH = BM_GMODE

Returns:

DX Current mode bits

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_get_error vc_get_error

Purpose: Gets error handler.

Syntax:

```
void (far cdecl * far pascal vc_get_error(err_code,
                                stack) ) ();
short err_code;          /* bus event code */
char far * far *stack;   /* address of stack base of
                            current handler */
```

Description: Gets handlers for the various VXIbus error conditions, plus the watchdog timer. Error handling works like interrupt handling, with two exceptions:

1) Where the interrupt handler is passed the Status/ID of the VXIbus interrupter, the error handler is passed the error number (see list below).

2) If the error is BERR, the BusManager clears BERR before calling the handler.

This command uses the error constants defined in **busmgr.h**. See `vc_set_error`.

Return Value: Entry address of current handler.

Errors: None.

Assembly: *Entry:*

AH	=	BM_GEH
AL		Bus event code (`err_code`)
	0	SYSFAIL
	1	Bus error (BERR)
	2	ACFAIL
	3	Watchdog timer

Returns:

ES:DI	Address of current driver assigned to interrupt
BX:SI	Address of stack base of current handler (`stack`)
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_get_intr vc_get_intr

Purpose: Returns the address of the current interrupt handler without changing it.

Syntax:

```
void (far cdelc * far pascal vc_get_intr( interrupt,
                                stack) ) ();
short interrupt;          /* VXIbus interrupt */
char far * far *stack;    /* address of stack base of current
                          handler */
```

Description The return value of `vc_get_intr` is the address of the current driver assigned to the interrupt. This is the value returned by the assembly language interface in ES:DI. If the second parameter is NULL, the stack base address is not returned.

Return Value: Address of current driver assigned to interrupt.

Errors: None

Assembly: *Entry:*

AH	= BM_GIH
AL	VXIbus interrupt (1..7), 0 for message interrupt, or 8 for Event/response pending `(interrupt)`

Returns:

ES:DI	Address of current driver assigned to interrupt
BX:SI	Address of stack base of current handler `(stack)`
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_get_slavebase

vc_get_slavebase

Purpose: Gets slave memory base address.

Syntax: `unsigned long far pascal vc_get_slavebase();`

Description: Returns the current base address of where VX5530/VX5535 memory shows up in A32 space on the VXIbus. Returns a **long** (32-bit) `ERR_FAIL` (i.e., -1L) if slave memory is disabled.

Return Value: Current slave memory base address in A32 space.

Errors: Returns `ERR_FAIL`, if slave memory is disabled.

Assembly: *Entry:*

AH	= BM_GSMB
AL	Reserved. Must be zero.

Returns:

AL	= 0 if slave memory disabled, = 1 if slave memory enabled
BX:DX	Current slave memory base address in A32 space, the most significant 16 bits in BX and the least significant 16 bits in DX

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_get_ula vc_get_ula

Purpose: Reports the current Unique Logical Address of the VX5530/VX5535.

Syntax: `short far pascal vc_get_ula();`

Description: The controller's logical address is a number in the range 00H to FFH.

Return Value: A number in the range 00H to FFH.

Errors: None.

Assembly: *Entry:*

AH	=	BM_GULA
AL		Reserved. Must be zero.

Returns:

DX	Previous base address for registers in A16 space
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_hw_ver

vc_hw_ver

Purpose: This function returns the version number for the hardware.

Syntax:

```
short far pascal vc_hw_ver();
```

Description: The hardware version number indicates the VX5530/VX5535 hardware that the BusManager application was created to run on.

Returns: Appropriate version number.

Errors: None.

Assembly: *Entry:*

AH =BM_VER

Returns:

BX Hardware Version Number

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_lws_cmd vc_lws_cmd

Purpose: Sends one longword serial command.

Syntax:

```
short far pascal vc_lws_cmd(ula, command, result_ptr, ticks);
unsigned short ula;      /* servant's unique logical addr */
unsigned long command;   /* command to send */
unsigned long far *result_ptr; /* addr of result, or NULL
                                  if no result expected */
unsigned short ticks;    /* timeout in 18.2 Hz ticks */
```

Description: A command will be sent only when the servant device's WRDY bit is set. If `result_ptr` is non-NULL, waits for a result and returns it in the location pointed to by `result_ptr`.

Return Value: Returns `VC_SUCCESS` if successful or a negative number on failure.

Errors: Returns one of the following:

`ERR_TIMEOUT`	TimeOut
`ERR_BERR`	Bus Error
`ERR_WS`	Word Serial Protocol Error
`ERR_RTIMEOUT`	TimeOut on Receive
`ERR_RBERR`	Bus Error on Receive

Assembly: *Entry:*

AH	= BM_LWS_CMD
AL	Servant's ULA (`ula`)
BX,DX	Command to send (`command`)
ES:DI	Address at which to store result -or- 0 if no result expected (`result_ptr`)
CX	Number of 18.2 Hz clock ticks to send the command (`ticks`)

Returns:

AL	Return code. One of
	`VC_SUCCESS`
	`ERR_TIMEOUT`
	`ERR_BERR`
	`ERR_WS`
	`ERR_RTIMEOUT`
	`ERR_RBERR`

BX,DX	Result value
CX	Number of clock ticks remaining
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_map_bus

vc_map_bus

Purpose: Maps VMEbus address to internal controller address.

Syntax:

```
char far * far pascal vc_map_bus(vxi_addr);
unsigned long vxi_addr;          /* VXIbus address to map to */
```

Description: Given a bus address, sets up the map base and returns a VX5530/VX5535 pointer to the space. Accesses to the bus may be made via this pointer. All accesses are done according to the current access mode. It is up to the user to handle the situation when a range of addresses goes beyond the 64KB-aligned bus window.

NOTE

Since the VXIbus window is 64KB in size and aligned on a 64KB boundary, the BusManager uses only BX to set the mapping and passes DX back to the caller. ES is set to the location of the VXIbus window. Such arrangement is not guaranteed on future implementations of the bus map hardware.

Return Value: A VX5530/VX5535 pointer to the VXIbus location.

Errors: Returns NULL if error occurs.

Assembly: *Entry:*

AH	=	BM_MAP
DX		Low 16 bits of VXIbus address to which to map (`vxi_addr`)
BX		High 16 bits of VXIbus address to which to map (`vxi_addr`)

Returns:

ES:DX	Pointer to VXIbus location
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_rest_state vc_rest_state

Purpose: Restores VMEbus access state.

Syntax:

```
short far pascal vc_rest_state(state_stash);
unsigned long far *state_stash; /* previously saved
                                   state in internal format */
```

Description: This function restores the access mode and bus window base that were previously saved via `vc_save_state`. No checking of this internal format is done. It does not affect interrupts.

Return Value: None.

Errors: None.

Assembly: *Entry:*

AH = BM_REST
ES:DI Pointer to previously saved state in internal format (`state_stash`)

Returns:

None

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_save_state vc_save_state

Purpose: Saves access mode and bus window base as the VMEbus access state.

Syntax:

```
void far pascal vc_save_state(state_stash);
unsigned long far *state_stash;      /* 4 byte area to save
                                        mapping state */
```

Description: This function preserves access mode and bus window base in caller-supplied area. Saved information is in internal format, suitable only for a subsequent command, `vc_rest_state`, and not to be changed by the caller. This saved state does not include interrupts.

NOTE

The BusManager state is, in effect, part of the context of any program, interrupt handler, or TSR task that uses the bus. See notes on Bus-Manager Context for considerations in saving and restoring this context.

`vc_save_state` and `vc_rest_state` are much faster operations than `vc_set_acc_mode` and `vc_get_acc_mode`, and should be used if the information is not being changed.

Return Value: None

Errors: None

Assembly: Entry:

AH = BM_SAVE
ES:DI Pointer to a four byte area in which to save the mapping state (`state_stash`)

Returns:

Buffer at ES:DI contains internal format mapping state as in `state_stash`.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

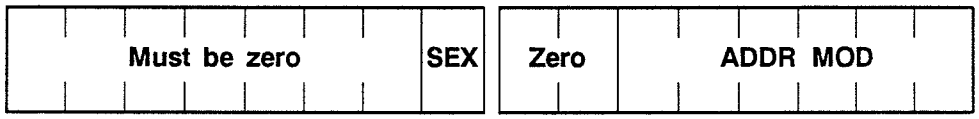
vc_set_acc_mode vc_set_acc_mode

Purpose: Sets the bus access parameters used on all subsequent operations.

Syntax:

```
short far pascal vc_set_acc_mode(newmode);
short newmode;     /* new mode bits */
```

Description: The new mode bit format is.



Where:

SEX	= 0 for Little Endian (386-style)
	= 1 for Big Endian (68000-style)
ADDR MOD	VXIbus address modifier

The hardware does not allow setting certain undefined VXIbus address modes. Setting undefined VXIbus address modes is an error and ERR_FAIL is returned.

The include file **busmgr.h** defines the following constants:

```
/*
 *  VXI Address Modifier Codes
 */

# define A16N 0x29      /* A16 non-privileged */
# define A16U A16N
# define A16S 0x2d      /* A16 supervisor */
# define A24ND 0x39     /* A24 non-privileged data */
# define A24NP 0x3a     /* A24 non-privileged program */
# define A24N A24ND
# define A24U A24ND
# define A24SD 0x3d     /* A24 supervisor data */
# define A24SP 0x3e     /* A24 supervisor program */
# define A24S A24SD
# define A32ND 0x09     /* A32 non-privileged data */
# define A32NP 0x0a     /* A32 non-privileged program */
# define A32N A32ND
# define A32U A32ND
# define A32SD 0x0d     /* A32 supervisor */
# define A32SP 0x0e     /* A32 supervisor */
# define A32S A32SD
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
/*
 *  VXI Byte Ordering bit to be or'ed in with above addr
    mod
 */

# define BM_MBO  (0x100)      /* Motorola byte ordering */
# define BM_IBO  (0)          /* Intel byte ordering */
```

Return Value: Returns `VC_SUCCESS` on no error or one of the following error codes.

Errors: Returns `ERR_FAIL` if the address mode is undefined.

Assembly: *Entry:*

AH = BM_SMODE
DX New mode bits (`newmode`)

Returns:

Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_set_error vc_set_error

Purpose: Defines error handlers.

Syntax:

```
void (far cdecl * far pascal vc_set_error(err_code,
                  new_handler, newstack, prev_stack) ) ();
short err_code;                 /* bus event code */
void (far cdecl * new_handler ) (); /* address of new
                                handler to assign to error */
char far *new_stack;            /* base of handler stack */
char far * far *prev_stack;     /* base of previous handler
                                stack */
```

Description: This function defines handlers for the various VXIbus error conditions, plus the watchdog timer. Error handling works just like interrupt handling, with two exceptions:

1) Where the interrupt handler is passed the Status/ID of the VXIbus interrupter, the error handler is passed the error number (see list below).

2) If the error is BERR, the BusManager clears BERR before calling the handler.

The following constants are defined in **busmgr.h**.

```
/*
 *  Error Constants
 */

# define BM_SYSFAIL_ERR                (0)
# define BM_BERR_ERR                   (1)
# define BM_ACFAIL_ERR                 (2)
# define BM_WATCHDOG_ERR               (3)
```

NOTE

The C calling convention is used to call error handling routines.

Return Value: Address of previous handler assigned to error.

Errors: None.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Assembly: Entry:

AH = BM_SEH
AL Bus event code (err_code)
0 SysFail
1 Bus error (BERR)
2 ACFAIL
3 Watchdog timer
ES:DI Address of new handler to assign to error (new_handler)
BX:SI Base of handler stack (new_stack)

Returns:

ES:DI Address of previous handler assigned to error
BX:SI Base of previous handler stack (prev_stack)
Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_set_intr

vc_set_intr

Purpose: This connects a bus interrupt to an interrupt routine on the VX5530/VX5535.

Syntax:

```
void (far cdecl * far pascal vc_set_intr( interrupt,
                    new_handler, new_stack, prev_stack) ) ();
short interrupt;                /* VXIbus interrupt */
void (far cdecl * new_handler ) (); /* address of new handler
                                       to assign to interrupt */
char far *new_stack;            /* base of handler stack */
char far * far *prev_stack;     /* base of previous handler
                                   stack */
```

Description: If an interrupt is not connected to a routine, it is masked off. If this function is called on an interrupt that is masked off, then the address of an internal do-nothing function is returned for the address of the previous interrupt handler. Setting an address of 0000:0000H masks off the interrupt.

`interrupt` specifies the interrupt for which a handler is to be set. `new_handler` is a **far** pointer to the interrupt handler to be assigned to the specified interrupt. `new_stack` is a stack pre-allocated by the user, that will be used when the `new_handler` routine is invoked.

The `new_handler` interrupt handler will be called when the associated interrupt occurs. The handler will be called using the `new_stack` specified, with a 32-bit Status/ID parameter using C calling conventions.

NOTE

The new stack pointer, `new_stack`*, should be a pointer to the high address of the block of memory that has been allocated for the stack, since stacks are built from high addresses to low addresses.*

e.g.,

Given an interrupt handler stack allocated as follows:

```
char MyStack[256];          /* interrupt stack */
```

The `new_stack` *parameter specifying this stack is:*

```
&MyStack[256]
```

It is an error to attempt to set an interrupt handler for an interrupt that has been redirected via `vc_er_redir()`. `vc_set_intr()` will return a NULL when such an operation is attempted.

The return value of `vc_set_intr` is the address of the previous driver assigned to interrupt. This is the value returned by the assembly language interface in ES:DI. The previous stack pointer is returned in `prev_stack`. If the fourth parameter is NULL, then the previous stack poiinter is not returned.

The following actions are taken when one of the specified interrupts occurs:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1) If this is a VXIbus interrupt, it is acknowledged. If it is a message interrupt, it is disabled. (Message interrupts are enabled by the message passing functions, described later in this manual.)

2) The bus state (mode and window base) is pushed on the stack. This is done here rather than by the interrupt routine for speed.

3) The BusManager switches to the handler's stack.

4) The 16-bit Status/ID value from the interrupt acknowledge is zero-extended to a **long** and pushed on the stack, high order word first.

NOTE

A 16-bit Status/ID is always requested. It is up to the handler to know the actual size of the Status/ID that the device returns.

5) The interrupt handler is entered via a FAR CALL. Interrupts are disabled upon entry to the handler (except for the Event/Response handler) but may be reenabled by the handler. In the case of the Event/Response handler, interrupts are enabled when the handler is called.

NOTE

Since the handler is entered from the BusManager via a FAR CALL, the handler should return to the BusManager with a RET instruction, and the BusManager will then clean up the stack.

The C calling convention is used.

6) The BusManager switches back to its own stack, the bus state is restored from the saved value on the stack, and interrupts are reenabled.

Return Value: Address of previous handler assigned to interrupt.

Errors: Returns NULL on error.

Assembly: *Entry:*

AH	= BM_SIH
AL	VXIbus interrupt (1..7), 0 for message interrupt, or 8 for Event/response pending `(interrupt)`
ES:DI	Address of new handler to assign to interrupt -or- 0000:0000H `(new_handler)`
BX:SI	Base of handler stack `(new_stack)`

Returns:

ES:DI	Address of previous handler assigned to interrupt
BX:SI	Base of previous handler stack `(prev_stack)`
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_sig_intr vc_sig_intr

Purpose: Signals a bus interrupt.

Syntax:

```
short far pascal vc_sig_intr(interrupt);
short interrupt;                    /* VXIbus interrupt */
```

Description: If `(interrupt)` is non-zero and the VX5530/VX5535 is not currently asserting an interrupt, then the appropriate bus interrupt (1 thru 7 on VXIbus) is signaled. If `(interrupt)` is non-zero and the VX5530/VX5535 *is* currently asserting an interrupt waiting for an acknowledgement, then an error is returned. If `(interrupt)` is zero and the VX5530/VX5535 is currently asserting an interrupt waiting for an acknowledgement, then the bus interrupt is deasserted. It is not an error to cancel an interrupt when no interrupt is asserted—this function always suceeds if `(interrupt)` is zero.

NOTE

Deasserting an interrupt without waiting for acknowledgement may cause certain hardware configurations to lock up and should be used with extreme care.

Return Value: Returns `VC_SUCCESS` for success.

Errors: Returns `ERR_FAIL` for failure.

Assembly: *Entry:*

AH = BM_SIG
AL VXIbus interrupt (1..7) *or* 0 to deassert interrupt `(interrupt)`

Returns:

Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_to_vme vc_to_vme

Purpose: Copies data from VX5530/VX5535 controller to VMEbus space.

Syntax:

```
unsigned short far pascal vc_to_vme(width, source, dest,
                                    count);
short width;              /* copy width */
char far *source;         /* start source address in VX5530
                          memory */
unsigned long dest;       /* VXIbus address */
unsigned short count;     /* byte count to transfer */
```

Description: Efficiently copies a block of memory from VX5530/VX5535 memory to consecutive bus locations, using the current access mode. This is intended for communicating between two VX5530/VX5535s, or dealing with devices such as network cards that map their memory onto the bus.

The `(width)` parameter specifies how the data is moved: whether data is to be moved in 8-, 16-, or 32-bit blocks, and whether error checking is performed (see below). VXIbus transfers are always done on natural boundary alignment—16-bit quantities are written to the VXIbus only at even addresses, and 32-bit quantities are written to the VXIbus only at addresses divisible by 4. Transfers to non-aligned locations are done in a read-modify-write fashion, with the destination block read, appropriate bytes copied to the block, and the block replaced.

For example, with a 32-bit copy to a non-aligned address, the leading 32-bit word would be read from the destination, modified, and written back; then whole 32-bit values would be transferred; finally the trailing 32-bit word would be read from the destination, modified, and replaced.

NOTE

The "read-modify-write" sequence is done in software, and is not a RMW bus cycle.

NOTE

If a non-modified byte in a leading or trailing word contains a semaphore, and that semaphore is signaled while the copy is taking place, that signal may be lost.

If the Fast Copy (C `define BM_FASTCOPY`) is not set in `(width)`, BERR is checked after every transfer. If there is an error, the copy is aborted and the BERR error handler is not called. This eliminates the need of requiring the program calling this function to coordinate with the BERR handler. Any errors are reflected by a non-zero value of this function.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

If the Fast Copy bit *is* set in `(width)`, only transfers to non-aligned locations (if any) are checked. This option takes advantage of the pipelining in the processor and the VXIbus interface hardware to achieve a five times improvement in transfer rate; however, errors may go undetected or the BERR error handler may be called (if the last transfer was still in the pipeline when the function returned and caused a BERR). In general, you should set the Fast Copy bit only if you don't expect a BERR and/or you don't care if you get one.

Note that `(count)` always contains the size of the transfer *in bytes* regardless of the width of the transfer. If `(count)` = 0, nothing will be transferred.

Any VXIbus mapping in place when this function is called is restored on return.

The following manifest constants are defined in **busmgr.h** to make life easier:

```
/*
*   Copy defines
*/
# define BM_W8          (1)         /* 8-bit xfer width */
# define BM_W16         (2)         /* 16-bit xfer width */
# define BM_W32         (4)         /* 32-bit xfer width */
# define BM_FASTCOPY    (0x80)      /* fast copy bit */
```

Return Value: Returns `VC_SUCCESS` if the copy is successful, otherwise it's the number of bytes remaining when a bus error occurs. This is the same as CX returned in the assembly language model.

Errors: Any errors are reflected by a non-zero return value.

Assembly: *Entry:*

AH	= BM_CPOUT
AL	Copy width `(width)`
	1 8-bit wide copy
	2 16-bit wide copy
	4 32-bit wide copy
	Optionally ORed with
	80H Fast Copy bit
ES:DI	Starting source address in VC memory `(source)`
BX	High 16-bits of VXIbus destination address `(dest)`
SI	Low 16-bits of VXIbus destination address `(dest)`
CX	Byte count to transfer, maximum of 64K-1 bytes `(count)`

Returns:

CX	= `VC_SUCCESS` if copy successful, otherwise number of bytes remaining on bus error
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_wait_intr
vc_wait_intr2

vc_wait_intr
vc_wait_intr2

Purpose: Waits for interrupt before executing caller.

Syntax:

```
short far pascal vc_wait_intr(mask, status, ticks);
unsigned short mask;            /* mask of interrupt */
unsigned long far *status;      /* returned status/ID */
unsigned short ticks;           /* # of 18.2 Hz clock ticks to
                                   wait for interrupt */

short vc_wait_intr2(mask, status, mem_ticks);
unsigned short mask;            /* mask of interrupt */
unsigned long far *status;      /* returned status/ID */
unsigned short far *mem_ticks; /* address of clock ticks */
```

Description: This function delays execution of the caller until either a bus interrupt arrives or a timer expires. The return code indicates which of these two happenstances transpired. In this implementation, only the highest priority interrupt is handled per call, where event/response pending is highest and the message interrupt is lowest. Other interrupts are left pending.

A 16-bit interrupt acknowledge cycle is performed when the interrupt arrives. It is up to the program to know whether the device returns an 8- or 16-bit Status/ID. For compatability with future products, this value is zero-extended to 32 bits.

`vc_wait_intr` and `vc_wait_intr2` return the mask of the highest priority interrupt that occurred (*e.g.*, `1 << intno`), zero if the timer expired, and ERR_FAIL if some other error occurred. `vc_wait_intr` and `vc_wait_ intr2` differ in that `vc_wait_intr` takes clock ticks as a parameter, while `vc_wait_intr2` takes a *pointer* to clock ticks as a parameter and returns the number of remaining clock ticks in that location.

The timer value is specified in DOS clock ticks (18.2Hz) . If `(ticks) =0`, only one check will be made before returning. The appropriate interrupt acknowledge is done before returning to the caller. No bus state saving or restoring is done.

If an interrupt also has a handler assigned to it (see `vc_set_intr`), then the handler is executed before returning.

Interrupts are "sticky" in that whenever an interrupt occurs, that fact is remembered and will be returned by `vc_wait_intr`. This eliminates the race condition that would otherwise exist between the device generating the interrupt and the program waiting for the interrupt; however, it can cause problems with the BusManager remembering stale interrupts. To avoid this problem a dummy wait with zero ticks should be done before using the device in order to clear out stale interrupts.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

NOTE

In order to use the clock for timing out, DOS interrupts are enabled.

Return Value: Mask of the interrupt that occurred or zero if timer expired.

Errors: Returns `ERR_FAIL` on error.

Assembly: *Entry:*

AH = BM_VXIWAIT
BX Mask of interrupts for which to wait `(mask)`
- 01H Message interrupt
- 02H VXIbus interrupt 1
- 04H VXIbus interrupt 2
- 08H VXIbus interrupt 3
- 10H VXIbus interrupt 4
- 20H VXIbus interrupt 5
- 40H VXIbus interrupt 6
- 80H VXIbus interrupt 7
- 100H Event/response pending

CX Number of 18.2Hz clock ticks to wait for interrupt `(ticks)`

Returns:

AX Mask of the interrupt that occurred -or- zero if timer expired
BX Zero (0)
DX Status/ID of the highest priority interrupt `(status)`
CX Number of clock ticks remaining when the interrupt arrived
Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_watchdog vc_watchdog

Purpose: Specifies watchdog timer functions.

Syntax:

```
short far pascal vc_watchdog(code);
short code;        /* watchdog timer subfunction code */
```

Description: The watchdog timer is designed to be used to protect against system lock-up in the case of error. If the timer is not reset or *tickled* within the specified interval, a watchdog error is signaled. Typically, the timer is tickled in frequently executed pieces of code, but never where there is the danger of infinite loops (*e.g.*, waiting for a device to become ready).

Two timer durations are available: 125 msec (suitable for hard realtime) and 8 sec.

A watchdog timer error handler is set via `vc_set_error`. The default handler disables the timer.

Return Value: Returns `VC_SUCCESS` if successful.

Errors: Returns `ERR_FAIL` on error.

Assembly: *Entry:*

AH	=	BM_WDT
AL		Subfunction code `(code)`
	0	Reset (tickle) timer
	1	Disable timer
	2	Enable timer
	3	Set fast (125 msec) mode
	4	Set slow (8 sec) mode

Returns:

Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_ws_cmd vc_ws_cmd

Purpose: Sends one word serial command.

Syntax:

```
short far pascal vc_ws_cmd(ula, command, result_ptr, ticks);
unsigned short ula;      /* servant's unique logical addr */
unsigned short command; /* command to send */
unsigned short far *result_ptr; /* addr of result, or
                                   NULL if no result expected*/
unsigned short ticks;    /* timeout in 18.2 Hz ticks */
```

Description: A command will be sent only when the servant device's WRDY bit is set.

If `result_ptr` is non-NULL, waits for a result and returns it in the location pointed to by `result_ptr`.

Return Value: Returns `VC_SUCCESS` if successful or a negative number on failure.

Errors: Returns one of the following:

`ERR_TIMEOUT`	TimeOut
`ERR_BERR`	Bus Error
`ERR_WS`	Word Serial Protocol Error
`ERR_RTIMEOUT`	TimeOut on Receive
`ERR_RBERR`	Bus Error on Receive

Assembly: *Entry:*

AH	= BM_WS_CMD
AL	Servant's ULA (`ula`)
BX	Command to send (`command`)
ES:DI	Address at which to store result -or- 0 if no result expected (`result_ptr`)
CX	Number of 18.2 Hz clock ticks to send the command (`ticks`)

Returns:

AL	Return code. One of
	`VC_SUCCESS`
	`ERR_TIMEOUT`
	`ERR_BERR`
	`ERR_WS`
	`ERR_RTIMEOUT`
	`ERR_RBERR`

BX	Result value
CX	Number of clock ticks remaining
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_ws_rcv_str vc_ws_rcv_str

Purpose: Receives a series of bytes via the word serial BYTE REQUEST command.

Syntax:

```
short far pascal vc_ws_rcv_str(ula, msg_ptr, len,
                                bytecnt_ptr, ticks);
unsigned short ula;      /* servant's unique logical addr */
char far *msg_ptr;       /* message buffer */
short len;               /* message buffer length */
short far *bytecnt_ptr; /* number of bytes received */
unsigned short ticks;    /* time-out in 18.2 Hz ticks */
```

Description: BYTE REQUEST commands are sent only when the device's DOR (Data Output Ready) and WRDY bits are set.

If `bytecnt_ptr` is non_NULL, returns the number of bytes received in the location pointed to by `bytecnt_ptr`.

This function will terminate successfully when a byte with the END bit set is received. It will also terminate when the buffer is full, when a timeout occurs, when a VXIbus error occurs, or when a Word Serial Protocol error is detected.

Return Value: Returns `VC_SUCCESS` if successful or a negative number on failure.

Errors: Returns one of the following:

`ERR_TIMEOUT`	TimeOut
`ERR_BERR`	Bus Error
`ERR_WS`	Word Serial Protocol Error
`ERR_BUFFER_FULL`	Buffer Full
`ERR_RTIMEOUT`	TimeOut on Receive
`ERR_RBERR`	Bus Error on Receive

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Assembly: *Entry:*

AH = BM_WS_RCVSTR
AL Servant's ULA (ula)
BX:SI Address of buffer to store received string (msg_ptr)
DX Length of receive buffer (len)
CX Number of 18.2 Hz clock ticks to send the string (ticks)

Returns:

AL Return code. One of
VC_SUCCESS
ERR_BUFFER_FULL
ERR_TIMEOUT
ERR_BERR
ERR_WS
ERR_RTIMEOUT
ERR_RBERR
DX Number of bytes received (bytecnt_ptr)
CX Number of clock ticks remaining, in assembly only
Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_ws_serv_arm vc_ws_serv_arm

Purpose: Arms the local VXIbus data register for the command receive function.

Syntax:

```
short far pascal vc_ws_serv_arm(code);
short code;        /* subfunction code */
```

Description: The local VXIbus data register must be armed in order to receive a command. Arming for command receive causes a VXIbus-readable bit to be set indicating that a command will be accepted. In addition, the message interrupt may be enabled that will inform the program when the command arrives.

Subfunction code: `(code)`

0	Disarm command receive
1	Arm command receive
2	Arm command receive and enable message interrupt
4	Force disarm command receive
5	Force arm command receive
6	Force arm command receive and enable message interrupt

Subfunctions 0, 1, and 2 obey the VX5530/VX5535 locking protocol allowing multiple masters to communicate to the same device. This protocol requires that the VXI RESPONSE register not be touched by masters unless they are going to send a command. In environments where this rule may not be obeyed, the FORCE flavor of the various subfunctions may be used.

Return Value: Returns `VC_SUCCESS` if successful.

Errors: Returns `ERR_FAIL` on error.

Assembly: *Entry:*

AH = BM_ARM

AL Subfunction code `(code)`

0	Disarm command receive
1	Arm command receive
2	Arm command receive and enable message interrupt
4	Force disarm command receive
5	Force arm command receive
6	Force arm command receive and enable message interrupt

Returns:

Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_ws_serv_peek
vc_ws_serv_peek2

vc_ws_serv_peek
vc_ws_serv_peek2

Purpose: Wait for a word serial command.

Syntax:

```
short far pascal vc_ws_serv_peek(command, ticks);
unsigned long far *command;     /* only first 2 bytes used in
                                   16-bit command */
unsigned short ticks;    /*number of 18.2 Hz clock ticks to
                           wait for interrupt */

short far pascal vc_ws_serv_peek2(command, mem_ticks);
unsigned long far * far *command;    /* only first 2 bytes
                                        used in 16-bit command */
unsigned short far *mem_ticks;       /*address of clock ticks */
```

Description: `vc_ws_serv_peek` does the same thing as `vc_ws_serv_rcv`, except that it does not remove the incoming command. The command stays available for subsequent `vc_ws_serv_peek` or `vc_ws_serv_rcv` operations.

`vc_ws_serv_peek` and `vc_ws_serv_peek2` differ in that `vc_ws_serv_peek` takes clock ticks as a parameter, while `vc_ws_serv_peek2` takes a *pointer* to clock ticks as a parameter and returns the number of remaining clock ticks in that location.

NOTE

In order to use the clock for timing out, DOS interrupts are enabled.

Return Value: Returns the size of the command in bytes.

Errors: Returns 0 if timer expired, and
`ERR_FAIL` if other error

Assembly: *Entry:*

AH	= BM_MPEEK
CX	Number of 18.2 Hz clock ticks to wait for command `(ticks/mem_ticks)`

Returns:

AL	Number of command bytes received (0, 2, or 4) (in assembly only)
DX, BX-DX	Received command (if any) . 16-bit commands go in DX, 32-bit commands go in BX:DX with the most significant 16 bits in BX and the least significant 16 bits in DX
CX	Time remaining when command arrived (zero if timer expired) `(mem_ticks)`
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_ws_serv_rcv
vc_ws_serv_rcv2

vc_ws_serv_rcv
vc_ws_serv_rcv2

Purpose: The VX5530/VX5535 as a servant waits for a command to arrive and returns it.

Syntax:

```
short far pascal vc_ws_serv_rcv(code, command, ticks);
short code;                          /* subfunction code */
unsigned long far *command;          /* only first 2 bytes used in
                                     16-bit command */
unsigned short ticks;    /*number of 18.2 Hz clock ticks to
                         wait for interrupt */

short far pascal vc_ws_serv_rcv2(code, command, mem_ticks);
short code;                          /* subfunction code */
unsigned long far * far *command;    /* only first 2 bytes used
                                     in 16-bit command */
unsigned short far *mem_ticks; /*address of clock ticks */
```

Description: `(ticks)` or the data buffer of `(mem_ticks)` contains the number of clock ticks for which to wait before giving up and returning an error. If `(ticks)` or the data buffer of `(mem_ticks)` = 0, only one attempt will be made. The high 16-bits of the command are undefined if a 16-bit command is received.

Subfunction code: operation after command received `(code)`

0	Disarm command receive
1	Arm command receive
2	Arm command receive and enable command interrupt

`vc_ws_serv_rcv` and `vc_ws_serv_rcv2` differ in that `vc_ws_serv_rcv` takes clock ticks as a parameter, while `vc_ws_serv_rcv2` takes a *pointer* to clock ticks as a parameter and returns the number of remaining clock ticks in that location.

NOTE

In order to use the clock for timing out, DOS interrupts are enabled.

Return Value: The size of the command received.

Errors: Returns 0 if timer expired, and
`ERR_FAIL` if other error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Assembly: *Entry:*

AH = BM_MRCV

AL Subfunction code: operation after command received (code)

- 0 Disarm command receive
- 1 Arm command receive
- 2 Arm command receive and enable command interrupt

CX Number of 18.2 Hz clock ticks to wait for command (ticks/mem_ticks)

Returns:

AL Number of command bytes received (0, 2, or 4) (in assembly only)

DX, BX-DX Received command (if any) . 16-bit commands go in DX, 32-bit command go in BX:DX with the most significant 16 bits in BX and the least significant 16 bits in DX

CX Time remaining when command arrived (zero if timer expired)—result value of mem_ticks

Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_ws_serv_send
vc_ws_serv_send2

vc_ws_serv_send
vc_ws_serv_send2

Purpose: Sends a word serial command from this servant to its commander.

Syntax:

```
short far pascal vc_ws_serv_send(code, command, ticks);
short code;                          /* subfunction code */
unsigned long far *command;          /* only first 2 bytes
                                        used in 16-bit command */
unsigned short ticks;    /*number of 18.2 Hz clock ticks to
                           wait for interrupt */

short far pascal vc_ws_serv_send2(code, command, mem_ticks);
short code;                          /* subfunction code */
unsigned long far *command;          /* only first 2 bytes used
                                        in 16-bit command */
unsigned short far *mem_ticks; /* address of clock ticks */
```

Description: Sends a word serial command to this servant's commander by loading it to the VXIbus data registers and setting the outgoing data flag; however, before the command is sent the VXIbus data register must be clear. (ticks) or (mem_ticks) contains the number of clock ticks to wait before giving up. If (ticks) or (mem_ticks) = 0, only one check is made before returning an error. If there is no error and the low order bit of (code) is set, then the message interrupt is enabled. The high 16-bits of the command are not used if a 16-bit command is received; however, they *are* fetched by the C interface routines.

Number of command bytes to put (2, or 4) (code)

0	Put no command, just wait for previous command to be read
1	Enable command interrupt and put no command
2	Put 16-bit command
3	Put 16-bit command and enable command interrupt
4	Put 32-bit command
5	Put 32-bit command and enable command interrupt

vc_ws_serv_send and vc_ws_serv_send2 differ in that vc_ws_serv_send takes clock ticks as a parameter, while vc_ws_serv_send2 takes a *pointer* to clock ticks as a parameter and returns the number of remaining clock ticks in that location.

NOTE

In order to use the clock for timing out, DOS interrupts are enabled.

Return Value: Returns VC_SUCCESS if successful.

Errors: Returns ERR_FAIL on error.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Assembly: *Entry:*

AH = BM_MPUT

AL Number of command bytes to put (2, or 4)) `(code)`

- 0 Put no command, just wait for previous command to be read
- 1 Enable command interrupt and put no command
- 2 Put 16-bit command
- 3 Put 16-bit command and enable command interrupt
- 4 Put 32-bit command
- 5 Put 32-bit command and enable command interrupt

DX, BX-DX Command to put. 16-bit commands go in DX, 32-bit commands go in BX:DX with the most significant 16 bits in BX and the least significant 16 bits in DX

CX Number of 18.2 Hz clock ticks to wait for VXIbus data register to clear `(ticks/mem_ticks)`

Returns:

CX Time remaining when VXIbus data register cleared (zero if timer expired) `(mem_ticks)`

Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_ws_snd_str

vc_ws_snd_str

Purpose: Sends a series of bytes via the word serial BYTE AVAILABLE command.

Syntax:

```
short far pascal vc_ws_snd_str(ula, msg_ptr, len,
                                bytecnt_ptr, ticks);
unsigned short ula;      /* servant's unique logical addr */
char far *msg_ptr;       /* addr of string to send */
short len;               /* message length */
short far *bytecnt_ptr; /* number of bytes sent */
unsigned short ticks;    /* time-out in 18.2 Hz ticks */
```

Description: BYTE AVAILABLE commands are sent only when the device's DIR (Data Input Ready) and WRDY bits are set. This function sets the END bit in the last command of the series.

If `bytecnt_ptr` is non_NULL, returns the number of bytes sent in the location pointed to by `bytecnt_ptr`.

Return Value: Returns `VC_SUCCESS` if successful or a negative number on failure.

Errors: Returns one of the following:

`ERR_TIMEOUT`	TimeOut
`ERR_BERR`	Bus Error
`ERR_WS`	Word Serial Protocol Error

Assembly: *Entry:*

AH	= BM_WS_SNDSTR
AL	Servant's ULA (`ula`)
BX:SI	Address of the string to send (`msg_ptr`)
DX	Message length (`len`)
CX	Number of 18.2Hz clock ticks to send the string (`ticks`)

Returns:

AL	Return code. One of
	`VC_SUCCESS`
	`ERR_TIMEOUT`
	`ERR_BERR`
	`ERR_WS`

DX	Number of bytes sent (`bytecnt_ptr`)
CX	Number of clock ticks remaining
Carry Flag	Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_ws_stat vc_ws_stat

Purpose: Returns the word serial status of the designated servant.

Syntax:

```
short far pascal vc_ws_stat(ula);
unsigned short ula;     /* servant's unique logical addr */
```

Description: The status includes bits 14-8 of the servant's Bus Response Register returned as bit 14-8 of the result. This includes a reserved bit (14), DOR, DIR, ERR*, Read Ready, Write Ready, and FHS*.

Return Value: Returns status bits 14-8 of the designated servant.

Errors: Returns `ERR_BERR` on error.

Assembly: *Entry:*

AH = BM_WS_STAT
AL Servant's ULA `(ula)`

Returns:

AL Bits 14 through 8 of device's Bus Response Register
or ERR_BERR
CX Number of clock ticks remaining, in assembly only
Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_vme_ctrl

vc_vme_ctrl

Purpose: Allows reading, setting, and clearing of various control bits in the VMEbus subset of the VXIbus interface.

Syntax:

```
short far vc_vme_ctrl(code, flag);
unsigned short code;     /* subfunction code */
unsigned short flag;     /* flag code */
```

Description: The values are non-zero for asserted and zero for deasserted, without regard for the polarity of the actual hardware bits. For instance, asserting EXTENDED Self Test actually causes a hardware control bit to be set to zero, indicating that the VX5530/VX5535 is executing an extended self test because the EXTENDED Self Test control bit is negative true logic. This method allows software to be insulated from the actual flavor of the hardware controls.

See the **Assembly** heading for description of subfunction and flag codes.

Return Value: The return value of the function is defined only for subfunction 0 (read) . It should be checked only for equality to zero—individual bit settings depend on the particular flag code.

Errors: Returns `ERR_FAIL` on error.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Assembly: *Entry:*

AH = BM_VCTRL

AL Subfunction code (`code`)

0	Read flag
1	Assert flag
2	Deassert flag

DL Flag code (`flag`)

0	SYSFAIL out enable
1	SYSRESET in enable
2	(reserved)
3	SYSRESET out
4	Self test PASS
5	EXTENDED Self Test
6*	Watchdog Timer Expired
7*	VME ACFAIL asserted
8†	VME BERR asserted
9*	VME SYSFAIL asserted
10*	VMEbus signal available
11*	VMEbus access (read or write)
12*	VMEbus write access
13*	VMEbus pipeline busy
14	Sticky BERR. Set whenever a BERR occurs and cleared by software. Note: BusManager clears this bit when it detects a BERR.

Flags marked with * are read-only.

†BERR may be read and deasserted, but not asserted.

Returns:

AL Defined for subfunction 0 (read) only:

= 0 if flag not asserted

≠ 0 if flag asserted

Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

vc_vxi_ctrl vc_vxi_ctrl

Purpose: Allows the reading, setting and clearing of various control bits in the VXIbus interface.

Syntax:

```
short far pascal vc_vxi_ctrl(code,flag);
unsigned short code;     /* Subfunction Code */
unsigned short flag;     /* Flag Code */
```

Description: Values are non-zero for asserted and zero for deasserted, without regard for the polarity of the actual hardware bits. The flag code for triggers is the same as returned by `ORLMAllocate` (see section on the On Line Resource Manager). Thus, an application may call `ORLMAllocate` to allocate a trigger, and pass that same value to `vc_vxi_ctrl` to use the trigger.

See the **Assembly** heading for description of subfunction and flag codes.

Return Value: If the code passed into the function is READ (0), a 1 (one) will be returned if the flag is asserted, and a 0 (zero) will indicate that the flag is deasserted. If the code passed into the function is ASSERT (1) or DEASSERT (2), then a `VC_SUCCESS` will be returned from the function if it is successful.

READing (0) a trigger returns the value that was last set via ASSERT or DEASSERT. SAMPLing (3) a trigger samples the actual value of the trigger line on the VXIbus. For a non-trigger flag (DIR, DOR, or ERR) READing and SAMPLing are equivalent.

Errors: Returns `ERR_FAIL` if attempting to access a trigger line not defined on this hardware.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Assembly: *Entry:*

AH = BM_VXI_CTRL

AL Subfunction code (code)

0	Read flag
1	Assert flag
2	Deassert flag
3	Sample trigger

DL Flag code (flag)

0	Data Input Ready (DIR)
1	Data Output Ready (DOR)
2	ERR flag
OLRM_TTLTRG0	TTL Trigger Line 0
OLRM_TTLTRG1	TTL Trigger Line 1
OLRM_TTLTRG2	TTL Trigger Line 2
OLRM_TTLTRG3	TTL Trigger Line 3
OLRM_TTLTRG4	TTL Trigger Line 4
OLRM_TTLTRG5	TTL Trigger Line 5
OLRM_TTLTRG6	TTL Trigger Line 6
OLRM_TTLTRG7	TTL Trigger Line 7
OLRM_ECLTRG0	ECL Trigger Line 0
OLRM_ECLTRG1	ECL Trigger Line 1
OLRM_ECLTRG2*	ECL Trigger Line 2
OLRM_ECLTRG3*	ECL Trigger Line 3
OLRM_ECLTRG4*	ECL Trigger Line 4
OLRM_ECLTRG5*	ECL Trigger Line 5

Trigger lines marked with an asterisk (*) are for future expansion and will not work on the standard VX5530/VX5535.

Returns:

AL Defined for subfunction 0 (read) only:
= 0 if flag not asserted
= 1 if flag asserted
= ERR_FAIL if trigger line is not defined for this hardware

Carry Flag Set if error

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Section 5
TekVICS Online Resource Manager

Introduction

The Online Resource Manager (OLRM) gives application programs a high-level attribute-type interface to the devices in the VXIbus, and manages serially reusable resources, such as interrupt and trigger lines. The OLRM also allows VMEbus devices to be viewed in the same way as VXIbus devices.

The OLRM is attribute oriented, and allows devices to be addressed by either symbolic device name or logical address.

This section describes each command in detail. Commands are located in alphabetical order.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Calling the OLRM From C

The following are examples of the usage of the OLRM functions.

```
i = OLRMGetNumAttr("Wavegen",0,OLRM_SLOT);
```

Get the slot number of device Wavegen

```
i = OLRMGetNumAttr("globalmem",0,OLRM_ADDRESS_BASE);
```

Get the memory base address of device globalmem

```
if (OLRMGetBoolAttr("scanner1",0,OLRM_SIGREG)) ...
```

Test scanner1 for a signal register

```
manufname = OLRMGetStringAttr("Wavegen",0,
 OLRM_MANUFACTURER, manufname);
```

Get symbolic manufacturer's name of Wavegen

```
OLRMGetList(NULL,0,OLRM_DEVICES,256,lalist);
```

Get list of logical addresses of all devices

```
OLRMRename(NULL,25,"Mil1553");
```

Rename the device with logical address 25 as Mil1553

```
i = OLRMAllocate(OLRM_TTLTRGANY2);
```

Allocate any two adjacent TTL trigger lines

olrm.h and **vc_err.h** are the two header files associated with the OLRM.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Alphabetical Command Listing

Following is an alphabetical listing and a brief purpose statement for each command. Commands are listed in detail in the remainder of this section.

OLRMAllocate	Allocates one, or a group of, trigger lines and interrupt lines
OLRMDeallocate	Deallocates one, or a group of, trigger lines and interrupt lines
OLRMGetBoolAttr	Returns a boolean attribute of a device
OLRMGetList	Returns an array of information about multiple devices
OLRMGetMedia	Returns a value indicating which bridge a device uses
OLRMGetNumAttr	Returns a numerical attribute of a device
OLRMGetSlotName	Returns an ASCII string for the name of the device in a particular slot
OLRMGetStringAttr	Returns an ASCII string attribute of a device
OLRMRename	Renames a device

For most of these commands, the first two parameters are an ASCII device name and a numeric logical address. One or the other is used to refer to the device. In the C interface, if the ASCII device name parameter is present (i.e., is not NULL), it is used, and the second parameter is ignored. If the ASCII device name is NULL, the second parameter is used.

Unless otherwise noted, inappropriate parameters cause meaningless results to be returned (e.g., asking for the memory speed of a register-based VXIbus device).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OLRMAllocate OLRMAllocate

Purpose: Allocates trigger and interrupt line resources.

Syntax:

```
unsigned short far pascal OLRMAllocate(resource);
unsigned short resource;        /* type of resource */
```

Description: Resources can be allocated specifically ("give me TTLTRG 4") and generically ("give me two TTL trigger lines").

The resource parameter may be one of the following:

OLRM_TTLTRG0	OLRM_ECLTRG1
OLRM_TTLTRG1	OLRM_ECLTRG2
OLRM_TTLTRG2	OLRM_ECLTRG3
OLRM_TTLTRG3	OLRM_ECLTRG4
OLRM_TTLTRG4	OLRM_ECLTRG5
OLRM_TTLTRG5	OLRM_ECLTRG01
OLRM_TTLTRG6	OLRM_ECLTRG23
OLRM_TTLTRG7	OLRM_ECLTRG45
OLRM_TTLTRG01	OLRM_ECLTRGANY
OLRM_TTLTRG23	OLRM_ECLTRGANY2
OLRM_TTLTRG45	OLRM_IRQ1
OLRM_TTLTRG67	OLRM_IRQ2
OLRM_TTLTRG0123	OLRM_IRQ3
OLRM_TTLTRG4567	OLRM_IRQ4
OLRM_TTLTRGANY	OLRM_IRQ5
OLRM_TTLTRGANY2	OLRM_IRQ6
OLRM_TTLTRGANY4	OLRM_IRQ7
OLRM_ECLTRG0	OLRM_IRQANY

One can request the allocation of specific resources, groups of resources (e.g., TTL triggers 0 and 1), and "any" resources. To provide for D-size systems, resources include the extra four ECL triggers on the P3 connector.

To permit computation with resources, the encodings are numerically relative to the lowest-numbered resource of a class. Thus, for instance, OLRM_TTLTRG1 is equal to OLRM_TTLTRG0 + 1, and OLRM_IRQ3 is equal to OLRM_IRQ1 + 2.

Since the OLRM_ECLTRGANY and OLRM_ECLTRGANY2 cases could allocate, conceptually, nonexistent ECL triggers 2-5 in a C-size system, one should avoid using these in a C-size system.

NOTE

All resources are not necessarily available for allocation when the system is initialized. Specifically, the SURM allocates interrupt lines for VMEbus devices as described by use of the Configurator.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Return Value: If the resource was allocated, the returned value is nonzero, and is the resource number. In the case of multiple allocations (e.g., `OLRMAllocate (OLRM_TTLTRGANY2)`), the value returned is that of the lowest-numbered of the resources. The returned value is 0 if the function is unsuccessful (e.g., specific resource already allocated, insufficient resources available, unknown resource).

Errors: None.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OLRMDeallocate

OLRMDeallocate

Purpose: Places the specified resource(s) in the deallocated (available for allocation) state.

Syntax:

```
void far pascal OLRMDeallocate(resource);
unsigned short resource;        /* type of resource */
```

Description: The resource parameters can be any of those for OLRMAllocate, except for the "any" values.

Return Value: None.

Errors: None.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OLRMGetBoolAttr

OLRMGetBoolAttr

Purpose: Returns requested information about a specific device.

Syntax:

```
unsigned short far pascal OLRMGetBoolAttr(devname, ula,
                                          attr);
char far *devname;          /* device name */
unsigned short ula;         /* unique logical address */
unsigned short attr;        /* attribute */
```

Description: The device can be addressed by symbolic name or logical address. The attribute may be one of the following. The comments indicate from where the information is obtained. Where a VXIbus register is indicated, a simulated register is used for VMEbus devices. The source "devtab" is the internal device table maintained by the SURM and OLRM.

Attribute	Source
OLRM_REGISTER_DEVICE	VXIbus ID register
OLRM_MEMORY_DEVICE	"
OLRM_EXTENDED_DEVICE	"
OLRM_MESSAGE_DEVICE	devtab
OLRM_A16_ONLY	VXIbus ID register
OLRM_A16_A24	"
OLRM_A16_A32	"
OLRM_A24A32_ENABLED	VXIbus status register
OLRM_MODID	"
OLRM_EXTENDED_TEST	"
OLRM_PASSED	"
OLRM_SUPVSR_ONLY	VXIbus memory attribute register
OLRM_BT	"
OLRM_N_P	"
OLRM_D32	"
OLRM_CMDR	VXIbus message protocol register
OLRM_SIGREG	"
OLRM_MASTER	"
OLRM_INTERRUPTER	"
OLRM_FHS	"
OLRM_SHMEM	"
OLRM_DOR	VXIbus message response register
OLRM_DIR	"
OLRM_ERR	"
OLRM_RRDY	"
OLRM_WRDY	"
OLRM_FHS_ACTIVE	"

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Attribute	Source
OLRM_LOCKED	"
OLRM_FAILED	devtab
OLRM_NOTVXI	"
OLRM_MEM_ALLOCATED	"
OLRM_EXISTS	"

If the device is a VXIbus device, acquisition of most of the attributes causes a VXIbus access.

NOTE

The boolean value returned is always of positive logic, independent of the polarity of the actual VXIbus-defined bit. For instance,the attribute OLRM_MODID returns TRUE if the device's MODID bit is 0; OLRM_N_P returns TRUE if a RAM device is nonvolatile or a ROM device electrically programmable.

Since most of the attributes are named the same way as in the VXIbus specification, familiarity with the naming in the VXIbus specification is assumed. The OLRM_ FAILED attribute denotes whether the SURM reported the device as failed and placed the device in the safe state. The OLRM_NOTVXI attribute denotes whether the device is not a VXIbus device. The OLRM_MEM_ALLOCATED attribute denotes whether address space for the device was reserved or allocated in the A24 or A32 address space. The OLRM_EXISTS attribute denotes whether the referenced device (by symbolic name or logical address) is a known device.

Return Value: Information about the specified device.

Errors: If an error occurs (e.g., nonexistent device), the value returned is 0 (FALSE).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OLRMGetList

OLRMGetList

Purpose: Returns a list of information as bytes in a character array, and a return value indicating the number of elements in the list.

Syntax:

```
unsigned short far pascal OLRMGetList(devname, ula, attr,
                                      size, list);
char far *devname;        /* device name */
unsigned short ula;       /* unique logical address */
unsigned short attr;      /* attribute */
unsigned short size;      /* maximum number of bytes in list */
char far *list;           /* list of information */
```

Description: The `size` argument specifies the maximum number of bytes returned in the list (the return value is not influenced by `size` and thus may be greater than `size`).

Attribute	Source
OLRM_DEVICES	devtab
OLRM_SERVANTS	devtab

If the attribute is OLRM_DEVICES, the `devname` and `ula` arguments are ignored. The logical addresses of all devices in the system (VXIbus and pseudo-VXIbus) are returned in the list.

If the attribute is OLRM_SERVANTS, the logical addresses of the servants of the specified device are returned in the list. The device can be addressed by symbolic name (devname) or logical address.

Return Value: The number of elements in the list.

Errors: If an error occurs, the returned value is 0.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OLRMGetMedia

OLRMGetMedia

Purpose: Finds the bridge that a device uses.

Syntax:

```
long far pascal OLRMGetMedia(devname);
char far *devname;        /* device name */
```

Description: This function returns the bridge media which the device is using. The constant definition for the bridge media is in the file **mds.h**.

Return Value: Returns the bridge media of the device specified in the `devname` parameter.

Errors: Returns `ERR_ILLEGAL_NAME` on error.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP