Installation & Reference Guide

DOC. REV. 6/27/2014

VersaAPI

VersaLogic Application Programming Interface

WWW.VERSALOGIC.COM

12100 SW Tualatin Road
Tualatin, OR 97062-7341
(503) 747-2261
Fax (971) 224-4708

Notice:

Although every effort has been made to ensure this document is error-free, VersaLogic makes no representations or warranties with respect to this product and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose.

VersaLogic reserves the right to revise this product and associated documentation at any time without obligation to notify anyone of such changes.

Product Revision

Revision 1.00 – Commercial release.

Support Page

The VersaAPI Support Page contains information and resources for this product including:

- Product Download

VersaTech KnowledgeBase

The VersaTech KnowledgeBase is a useful resource for resolving technical issues with VersaLogic products.

Technical Support

If you have further questions, contact VersaLogic Technical Support at (503) 747-2261 or at Support@VersaLogic.com.

Contents

Overview 1

Description

The VersaLogic Application Programming Interface (VersaAPI) is a shared library of API calls for reading and controlling on-board devices on certain VersaLogic products. In Microsoft Windows, VersaAPI is represented as a dynamically linked library interface plus associated header file, and under Linux as a shared library with an associated header file.

The API functions are defined in the supplied header file VL_OSALib.h. This header file must be included in your application to be able to call the API functions. Also, the supplied library file VL_OSALib.lib must be linked into your application. This will enable your application to access the API calls in the supplied Dynamic Link Library (DLL).

Your application must run at administrator level to be able to open the device driver that accesses the hardware. Without this step you will receive undefined results.

Supported VersaLogic Boards

VersaAPI supports the following VersaLogic boards and functions as of the publication date of this document.

Table 1: Supported VersaLogic Boards and Functions

Board	Function			
	ADC	DAC	DIO	Counter Timer
Copperhead (VL-EBX-41)	✓	✓	✓	✓
Iguana (VL-EPIC-25)	✓	✓	✓	✓
Komodo (VL-EPICs-36)	✓	✓	✓	—
Mamba (VL-EBX-37)	✓	✓	✓	✓
Newt (VL-EPIC-17)	✓	✓	✓	✓
SPX-1	✓	—	—	—
SPX-2	—	—	✓	—
SPX-4	—	✓	—	—
VL-MPEe-A1/A2	✓	—	✓	—
VL-MPEe-U2	—	—	✓	—

Operating Systems

VersaAPI supports the following operating systems:

- Windows 7 (32 bit)
- Windows XP SP3 (32 bit)
- Windows XPe
- WES 7 (32 bit)
- Debian Linux (32 bit)

Windows Installation

The following procedures are intended as guidelines for installing VersaAPI on Windows machines. They apply to both Windows 7 and Windows XP installations. Some modifications to the procedures may be required depending on the configuration of your system. Contact VersaLogic Technical Support if you encounter problems with the installation.

These procedures require the use of Microsoft Visual Studio. Always run Visual Studio as Administrator or the device driver will not load. You will likely need to install the VS runtime executables for Visual Studio in order to run any applications using the API functions.

SETUP

1. Install and configure your VersaLogic CPU board and any expansion boards (Mini PCIe or SPX) that the VersaAPI will access.
2. Download the VersaAPI package and extract its contents into a temporary directory. Take note of the location of the temporary directory for access during these instructions.

PACKAGE CONTENTS

- Vcredist_x86.exe – Visual Studio 2008 redistributable installation package
- VersaAPI ReadMe.html – Readme documentation for VersaAPI
- VL_OSALib.dll – Dynamic Link Library for the executable (EXE)
- VL_OSALib.lib – Library file for linking the DLL to the EXE when compiling
- VL_OSALib.h – Header file with all VersaAPI definitions
- VLDrive.inf – CPU board device driver installation file
- VLDrivep.inf – Mini PCIe module device driver installation file
- VLDrive.sys – CPU board device driver
- VLDrivep.sys – Mini PCIe module device driver

INSTALLING THE DRIVERS

Note: Warning messages may be encountered when installing the drivers. Ignore these messages and continue with the installation.

Note: These instructions are for installing drivers for VersaLogic CPU boards and the VL-MPEe-A1/A2 expansion board. To install drivers for the VL-MPEe-U2, go to the VL-MPEe-U2 Support Page.

1. Open the Windows Hardware Wizard.
    a. In Windows 7, type hdwwiz in the Start menu search bar.
    b. In Windows XP, double-click the Add New Hardware icon on the Control Panel.
2. Select the option to select the hardware install manually and click Next. (Do not select the option for Windows to search for hardware.)

3. When the list of devices appears, select Show All Devices and click Next.
4. Click Have Disk.
5. Click Browse.
6. Navigate to the folder where you saved the VersaAPI package.

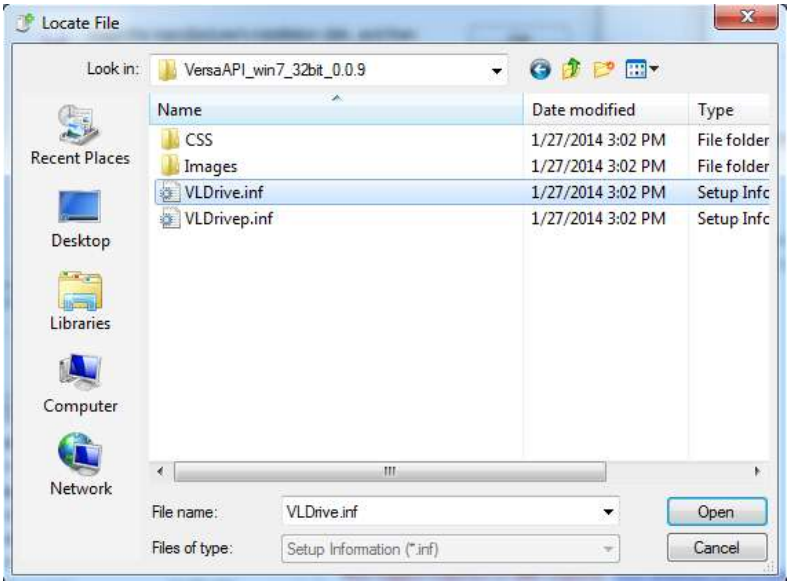

Figure 1. Selecting the VersaAPI Drivers

7. The setup information file (INF) you install depends on which VersaLogic boards and I/O functions you will be using:
   a. Select VLDrive.inf for VersaLogic CPU boards with on-board I/O or SPX expansion boards.
   b. Select VLDrivep.inf for VersaLogic the VL-MPEe-A1/A2 module.

   Note: If you will be using the I/O functions of both a CPU board and the Mini PCIe module, both drivers must be installed.
8. Click Open.
9. Click OK. You are given the option to install one of two drivers:
   - VersaLogic MPEe A1/A2 Driver
   - VersaLogic SPX Driver (KMDF)
10. Select the MPEe A1/A2 Driver or the VersaLogic SPX Driver (KMDF) for CPU board or SPX board functions.
11. Complete the onscreen instructions and restart the computer when prompted.

INCLUDING THE HEADER FILE

This procedure installs the VersaAPI header in your C++ program using the #include <VL_OSALib.h> call. This will load all VersaAPI program definitions.

1. In Visual Studio, create or open a C++ project.
2. Add a CPP file.
3. Select the Properties for the CPP file.
4. Expand the C/C++ section and select General.
5. In Additional Include Directories, enter or browse to the location of the VL_OSALIB.h header file.

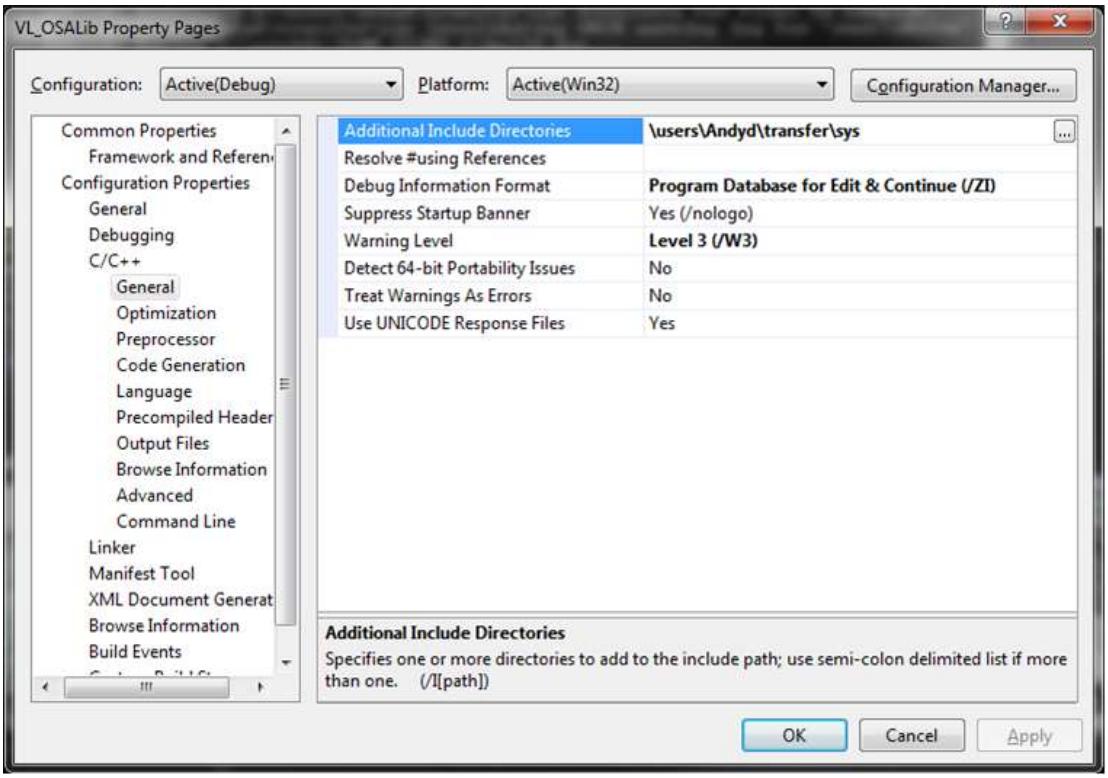

Figure 2. Including the Header File

6. Click OK. The header file has now been added to the project.

LINKING THE LIBRARY

1. In Visual Studio, open the Properties for your project.

2. In Project Properties > Configuration Properties > Linker, select Input.

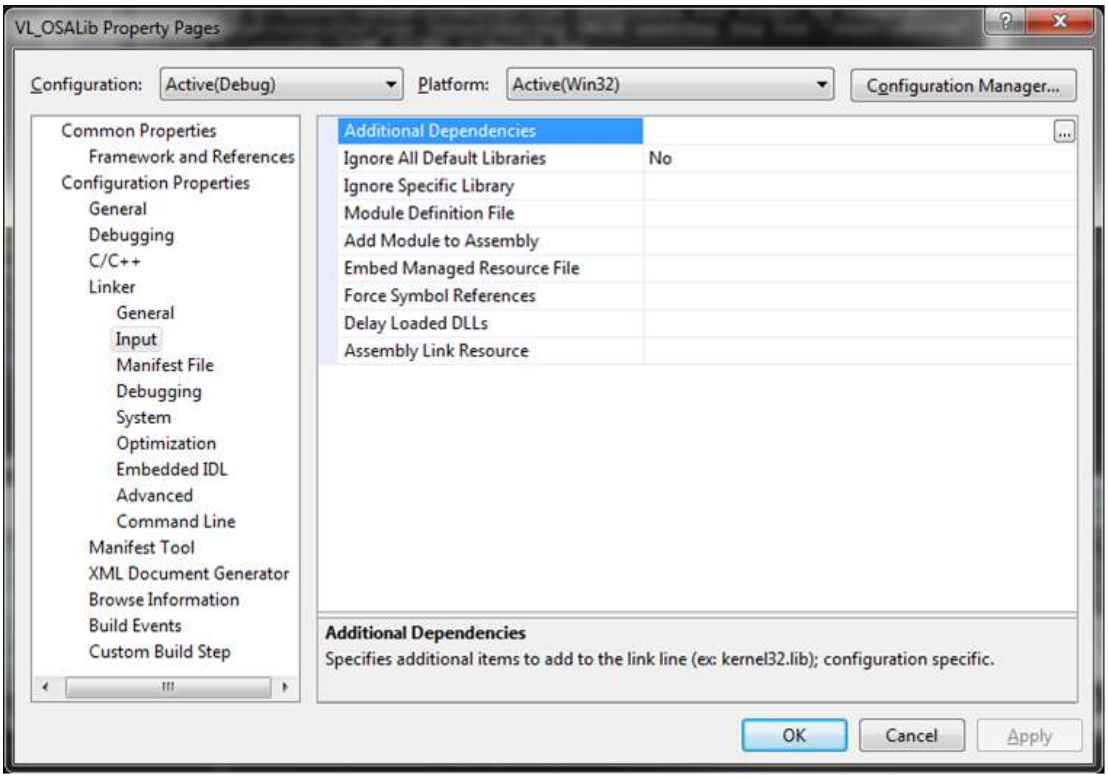

Figure 3. Selecting Additional Dependencies

3. In Additional Dependencies, enter vl_osalib.lib.

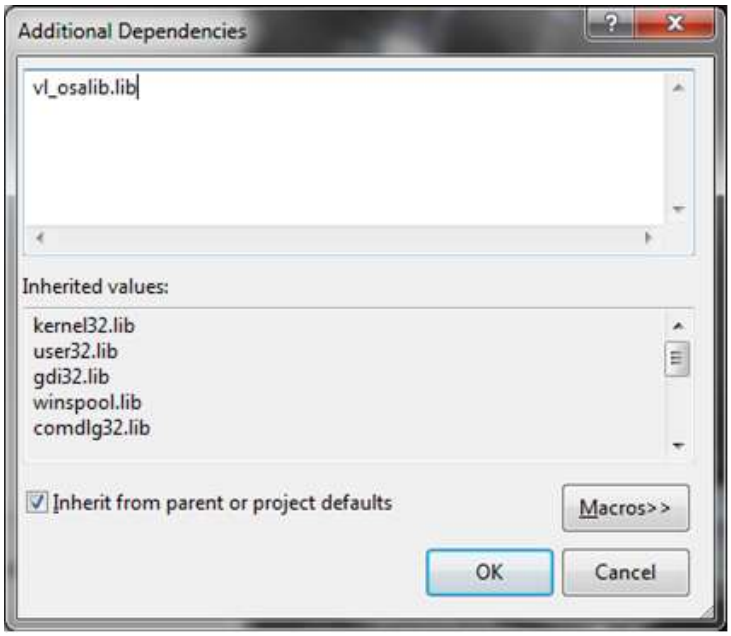

Figure 4. Linking the VersaAPI Library File

4. Now select Project Properties > Configuration Properties > Linker > General.

5. In Additional Library Directories, enter the location of the folder containing the library file and click OK. The library is now installed.

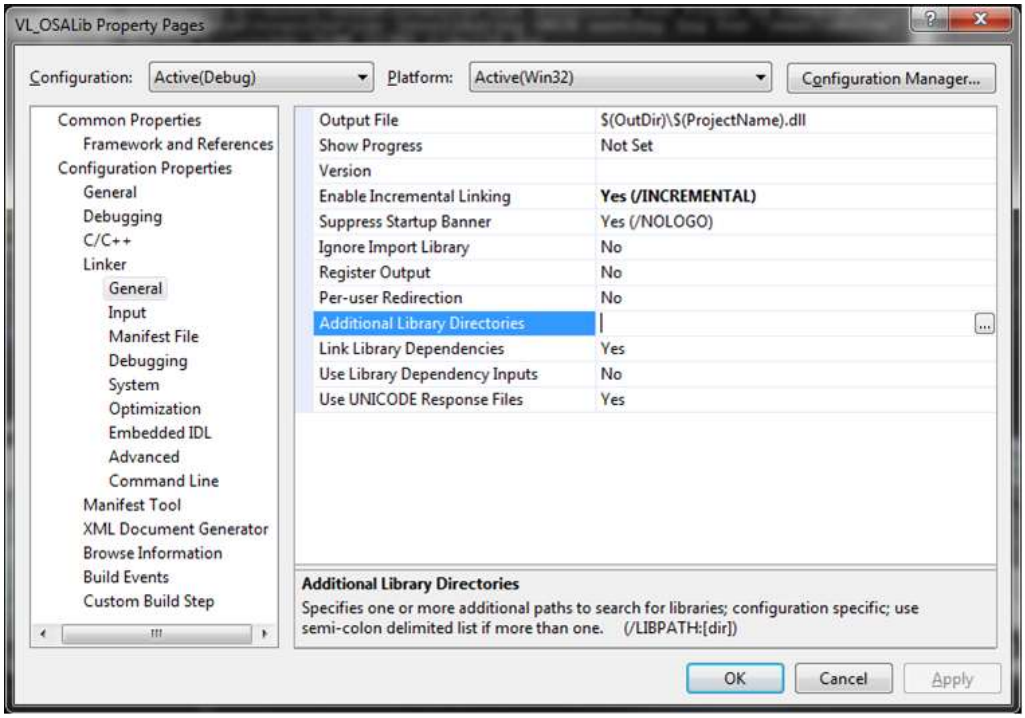

Figure 5. Adding the Library Folder

INCLUDING THE DLL IN THE PROJECT

The VL-OSALib.dll file must be copied into the debug and release folders for your Visual Studio project in order to utilize all VersaAPI functions. The DLL file must be in the same folder as the compiled EXE file.

Linux Installation

The VersaAPI package for Linux includes:

- **vl_install.sh** – An install script. To install the driver and shared library you must first change the permissions on the vl_install.sh within this package:

  ```
  chmod +x vl_install.sh
  ```

  and then run the installer using sudo (see the note below if using a kernel other than 3.2.0-4-686-pae):

  ```
  sudo ./vl_install.sh
  ```

  This installs the device drivers and the shared libraries.

- **libVL_OSALib.so** – A shared library file that contains the bulk of the VersaAPI library and affords access to VersaLogic CPU board and MiniPCIe I/O functions.

- **VL_OSALib.h** – A header file containing the necessary function prototypes for calling into the shared library, including definitions for particular commonly used values.

You can build applications using the VersaAPI shared library by including VL_OSALib.h in your source and "-lVL_OSALib" in your linker options.

Note: If you wish to use a different kernel than 3.2.0-4-686-pae, the driver source code has been provided in the src directory. For the build to succeed, you must have, at a minimum, already compiled the modules for your current kernel and installed the kernel header files on the build system. In each driver directory there is a makefile. Change to the relevant directory and issue the command:

```
make
```

on the command line, and the driver will build. The vldrivep directory is the pci vldrive version for use with the VL-MPEe-A1/A2 and the VL-MPEe-U2. The vldrive directory is the driver for VersaLogic CPU on board I/O resources, used for most I/O functions (DIO, ADC, DAC) on VersaLogic CPU boards supported by the library.

Once the drivers have built for your own kernel version, copy them into the same directory as the vl_install.sh script and then run:

```
sudo ./vl_install.sh
```

The VersaLogic drivers will now install automatically every time you boot the board.

Open and Close Calls

The library interface must be opened by every application that wishes to make calls into the API and also must be closed by that same application when exiting.

VL_Open();

Opens the VersaAPI library.

Syntax: VL_OSALIB_API unsigned long VL_Open();

Inputs: none

Outputs: unsigned long
This call returns 0 if the open was a success and nonzero if no useable drivers were found by the DLL.

VL_Close();

Closes the VersaAPI library.

Syntax: VL_OSALIB_API void VL_Close();

Inputs: none

Outputs: none

VL_GetVersion()

Gets the version number of the VersaAPI library

Syntax: VL_OSALIB_API void VL_GetVersion(unsigned char *Major, unsigned char *Minor, unsigned char *Revision);

Inputs: unsigned char *Major
A pointer to the unsigned character to receive the Version Major number.

unsigned char *Minor
A pointer to the unsigned character to receive the Version Minor number.

unsigned char *Revision
A pointer to the unsigned character to receive the Version Revision number.

Outputs: None.
While this function is void, the Major, Minor, and Revision versions are returned in their respective input fields.

Digital I/O (DIO) Calls

API calls can be made to control or interrogate DIO (also known as GPIO) channels.

DIO calls require you to identify the specific channel being accessed. See Digital I/O (DIO) Channel Assignments for channel definitions.

The following table lists the level and direction parameter definitions used in DIO calls.

Table 2: DIO API Parameter Definitions

Parameters	Value	
Level*	DIO_CHANNEL_LOW	0x00
	DIO_CHANNEL_HIGH	0x01
Direction	DIO_INPUT	0x01
	DIO_OUTPUT	0x00

*Level values are also the return results for VSL_DIOGetChannelLevel.

VL_DIOGetChannelLevel

Reads the signal level of the specified channel.

Syntax: VL_OSALIB_API unsigned char VL_DIOGetChannelLevel(unsigned char Channel);

Inputs: unsigned char Channel
The DIO channel number to be interrogated.

Outputs: unsigned char
Returns the state of the channel as either high (DIO_CHANNEL_HIGH) or low (DIO_CHANNEL_LOW).

VL_DIOSetChannelLevel

Sets the signal level of the specified channel.

Syntax: VL_OSALIB_API void VL_DIOSetChannelLevel(unsigned char Channel, unsigned char Level);

Inputs: unsigned char Channel
The DIO channel number to be set.

unsigned char Level
The DIO level to be set: DIO_CHANNEL_HIGH or DIO_CHANNEL_LOW.

Outputs: None.

VL_DIOSetChannelDirection

Sets the signal direction of the specified channel.

Syntax: VL_OSALIB_API void VL_DIOSetChannelDirection(unsigned char Channel, unsigned char Direction);

Inputs: unsigned char Channel
The DIO channel number to be set.

unsigned char Direction
The DIO direction to be set: DIO_INPUT or DIO_OUTPUT.

Outputs: None.

Analog-to-Digital Conversion (ADC) Calls

API calls can be made to control analog input channels.

ADC calls require you to identify the specific channel being accessed. See Analog to Digital (ADC) Channel Assignments for channel definitions.

The following table lists the range and format parameter definitions used in ADC calls.

Table 3: ADC API Parameter Definitions

Parameters	Value	
Range	SPI_RANGE_PM5V	0x00
	SPI_RANGE_PM10V	0x04
	SPI_RANGE_0_5V	0x08
	SPI_RANGE_0_10V	0x0C
Format	ADC_RAW	0x00
	ADC_VOLTS	0x01

VL_ADCSetAnalogInputRange

Sets the analog input channel and range for a selected channel.

Syntax: VL_OSALIB_API void VL_ADCSetAnalogInputRange(unsigned char Channel, unsigned char Range);

Inputs: unsigned char Channel
The analog input channel.

unsigned char Range
The analog input range for the channel.

Outputs: void

VL_ADCGetAnalogInput

Returns the value of the selected analog input channel.

Syntax: VL_OSALIB_API double VL_ADCGetAnalogInput(unsigned char Channel, unsigned char Format);

Inputs: unsigned char Channel
The analog input channel to be read.

unsigned char Format
How the data is returned. Either in its raw format or formatted into volts.

Outputs: double
The value read from the selected analog channel. This value will resolve to either 12 or 16 bits of accuracy depending on the accuracy of the channel.

VL_GetADCType

Returns the model of Analog Input card installed.

Syntax: VL_OSALIB_API unsigned char VL_GetADCType();

Inputs: void

Outputs: unsigned char
The type of VL-MPEe-Ax card: Either VSL_ADC_TYPE_A1 or VSL_ADC_TYPE_A2. If no VL-MPEe-Ax card is present, then the call will return 0.

Digital-to-Analog Conversion (DAC) Calls

API calls can be made to control analog output channels.

DAC calls require you to identify the specific channel being accessed. See Digital to Analog (DAC) Channel Assignments for channel definitions.

VL_ADCSetAnalogInputRange

Sets the analog input channel and range for a selected channel.

Syntax: VL_OSALIB_API void VL_DACSetAnalogOutput(unsigned char Channel, unsigned short Level);

Inputs: unsigned char Channel
One of the DAC channels (`SPI_AO_CHANNEL_1 - SPI_AO_CHANNEL_8`).

unsigned short Level
The value to set (between 0 and 0x0FFF = 0 and 4095 mV).

Outputs: None.

8254 Timer Calls

VersaLogic timers are based on emulated support for the Intel 8254 timer interface. See the Intel 8254/82C54: Introduction to Programmable Interval Timer page for details that will help you use the VersaAPI timer calls. More information on 8254 timer calls is readily found by searching the Internet.

The VersaAPI simplifies access to the 8254 timers into just four API calls.

VL_TMRSet

This call is used to start an 8254 timer. You must select a mode and a duration. It is recommended that you use the internal timer type, setting External to false.

Syntax: VL_OSALIB_API char VL_TMRSet(unsigned char Timer, unsigned char Mode, unsigned short Count, BOOL External);

Inputs: unsigned char Timer
One of the three timers (0-2).

unsigned char Mode
The 8254 timer mode to set (0-6).

unsigned short Count
The length of the timer. (For internal timers, the frequency is 4.125MHz.)

Outputs: Char
-1 if the timer could not be set, 0 if successful.

VL_TMRGet

Returns the current count of the timer.

Syntax: VL_OSALIB_API unsigned short VL_TMRSet(unsigned char Timer);

Inputs: unsigned char Timer
One of the three timers (0-2).

Outputs: Short
The current count of the specified timer.

VL_TMRClear

Stops the internal timer from counting down.

Syntax: VL_OSALIB_API void VL_TMRClear(unsigned char Timer);

Inputs: unsigned char Timer
One of the three timers (0-2).

Outputs: None.

VL_TMRGetEvent

Windows

This call will return an unsigned int that is actually a handle to an object that can be used in a wait type function in Windows. The object will become signaled when the timer in question has expired.

Syntax: VL_OSALIB_API unsigned int VL_TMRGetEvent(unsigned char Timer)

In Windows, the object type will be a handle to an event that can be used in a wait call using the WaitForSingleObject Windows API call.

Linux

In Linux you must provide a signal handler, and the VersaAPI driver will signal the handler that the interrupt for your timer has been asserted. The argument for the handler will be an unsigned integer that is a mask for the three timers.

Syntax: VL_OSALIB_API unsigned int VL_TMRRegisterHandler(void * HandlerRoutine)

Your routine would then be defined as:

```
#include <signal.h>

Void TMRSignalHandler(int Signal, siginfo_t *Interrupt, void *Nothing)
```

To determine which timer caused the interrupt, check the **Interrupt->si_int** field in your handler. The contents of Interrupt->si_int are defined as a bit mask, as shown below:

0x00000001 means Timer 0 has expired
0x00000002 means Timer 1 has expired
0x00000004 means Timer 2 has expired

Channel Assignment Tables

The following tables list the digital I/O (DIO), analog to digital conversion (ADC), and digital to analog conversion (DAC) channels found in the VersaAPI header file. The tables also list the connector pins used by each channel assignment on a number of VersaLogic boards.

Digital I/O (DIO) Channel Assignments

The definitions below are organized by board type and channel number:

- CPU Board On-Board DIO Channels
- VL-SPX-2 Expansion Board DIO Channels
- PCIe Expansion Board DIO Channels for the VL-MPEe-A1/2 and VL-MPEe-U2

CPU Board On-board DIO Channels

Channel Number	Hex Code	Channel Name	EBX-37 J17 Pin	EBX-41 J21 Pin	EPIC-17 J9 Pin	EPIC-25 J25 Pin
0	00	DIO_CHANNEL_1	1	1	1	16
1	01	DIO_CHANNEL_2	2	2	2	17
2	02	DIO_CHANNEL_3	3	3	3	18
3	03	DIO_CHANNEL_4	4	4	4	19
4	04	DIO_CHANNEL_5	6	6	6	21
5	05	DIO_CHANNEL_6	7	7	7	22
6	06	DIO_CHANNEL_7	8	8	8	23
7	07	DIO_CHANNEL_8	9	9	9	24
8	08	DIO_CHANNEL_9	11	11	11	26
9	09	DIO_CHANNEL_10	12	12	12	27
10	0A	DIO_CHANNEL_11	13	13	13	28
11	0B	DIO_CHANNEL_12	14	14	14	29
12	0C	DIO_CHANNEL_13	16	16	16	31
13	0D	DIO_CHANNEL_14	17	17	17	32
14	0E	DIO_CHANNEL_15	18	18	18	33
15	0F	DIO_CHANNEL_16	19	19	19	34
16	10	DIO_CHANNEL_17	21	21	21	—
17	11	DIO_CHANNEL_18	22	22	22	—
18	12	DIO_CHANNEL_19	23	23	23	—
19	13	DIO_CHANNEL_20	24	24	24	—
20	14	DIO_CHANNEL_21	26	26	26	—
21	15	DIO_CHANNEL_22	27	27	27	—
22	16	DIO_CHANNEL_23	28	28	28	—
23	17	DIO_CHANNEL_24	29	29	29	—
24	18	DIO_CHANNEL_25	31	31	31	—
25	19	DIO_CHANNEL_26	32	32	32	—
26	1A	DIO_CHANNEL_27	33	33	33	—
27	1B	DIO_CHANNEL_28	34	34	34	—
28	1C	DIO_CHANNEL_29	36	36	36	—
29	1D	DIO_CHANNEL_30	37	37	37	—
30	1E	DIO_CHANNEL_31	38	38	38	—
31	1F	DIO_CHANNEL_32	39	39	39	—

VL-SPX-2 EXPANSION BOARD DIO CHANNELS – USING SLAVE SELECT 0

Channel Number	Hex Code	Channel Name	VL-SPX-2 J2 Pin
64	40	DIO_SS0_CHANNEL_0	1
65	41	DIO_SS0_CHANNEL_1	2
66	42	DIO_SS0_CHANNEL_2	3
67	43	DIO_SS0_CHANNEL_3	4
			J3 Pin
68	44	DIO_SS0_CHANNEL_4	1
69	45	DIO_SS0_CHANNEL_5	2
70	46	DIO_SS0_CHANNEL_6	3
71	47	DIO_SS0_CHANNEL_7	4
			J4 Pin
72	48	DIO_SS0_CHANNEL_8	1
73	49	DIO_SS0_CHANNEL_9	2
74	4A	DIO_SS0_CHANNEL_10	3
75	4B	DIO_SS0_CHANNEL_11	4
			J5 Pin
76	4C	DIO_SS0_CHANNEL_12	1
77	4D	DIO_SS0_CHANNEL_13	2
78	4E	DIO_SS0_CHANNEL_14	3
79	4F	DIO_SS0_CHANNEL_15	4

VL-SPX-2 EXPANSION BOARD DIO CHANNELS – USING SLAVE SELECT 1

Channel Number	Hex Code	Channel Name	VL-SPX-2 J2 Pin
80	50	DIO_SS0_CHANNEL_0	1
81	51	DIO_SS0_CHANNEL_1	2
82	52	DIO_SS0_CHANNEL_2	3
83	53	DIO_SS0_CHANNEL_3	4
			J3 Pin
84	54	DIO_SS0_CHANNEL_4	1
85	55	DIO_SS0_CHANNEL_5	2
86	56	DIO_SS0_CHANNEL_6	3
87	57	DIO_SS0_CHANNEL_7	4
			J4 Pin
88	58	DIO_SS0_CHANNEL_8	1
89	59	DIO_SS0_CHANNEL_9	2
90	5A	DIO_SS0_CHANNEL_10	3
91	5B	DIO_SS0_CHANNEL_11	4
			J5 Pin
92	5C	DIO_SS0_CHANNEL_12	1
93	5D	DIO_SS0_CHANNEL_13	2
94	5E	DIO_SS0_CHANNEL_14	3
95	5F	DIO_SS0_CHANNEL_15	4

VL-SPX-2 EXPANSION BOARD DIO CHANNELS – USING SLAVE SELECT 2

Channel Number	Hex Code	Channel Name	VL-SPX-2 J2 Pin
96	60	DIO_SS0_CHANNEL_0	1
97	61	DIO_SS0_CHANNEL_1	2
98	62	DIO_SS0_CHANNEL_2	3
99	63	DIO_SS0_CHANNEL_3	4
			J3 Pin
100	64	DIO_SS0_CHANNEL_4	1
101	65	DIO_SS0_CHANNEL_5	2
102	66	DIO_SS0_CHANNEL_6	3
103	67	DIO_SS0_CHANNEL_7	4
			J4 Pin
104	68	DIO_SS0_CHANNEL_8	1
105	69	DIO_SS0_CHANNEL_9	2
106	6A	DIO_SS0_CHANNEL_10	3
107	6B	DIO_SS0_CHANNEL_11	4
			J5 Pin
108	6C	DIO_SS0_CHANNEL_12	1
109	6D	DIO_SS0_CHANNEL_13	2
110	6E	DIO_SS0_CHANNEL_14	3
111	6F	DIO_SS0_CHANNEL_15	4

VL-SPX-2 EXPANSION BOARD DIO CHANNELS – USING SLAVE SELECT 3

Channel Number	Hex Code	Channel Name	VL-SPX-2 J2 Pin
112	70	DIO_SS0_CHANNEL_0	1
113	71	DIO_SS0_CHANNEL_1	2
114	72	DIO_SS0_CHANNEL_2	3
115	73	DIO_SS0_CHANNEL_3	4
			J3 Pin
116	74	DIO_SS0_CHANNEL_4	1
117	75	DIO_SS0_CHANNEL_5	2
118	76	DIO_SS0_CHANNEL_6	3
119	77	DIO_SS0_CHANNEL_7	4
			J4 Pin
120	78	DIO_SS0_CHANNEL_8	1
121	79	DIO_SS0_CHANNEL_9	2
122	7A	DIO_SS0_CHANNEL_10	3
123	7B	DIO_SS0_CHANNEL_11	4
			J5 Pin
124	7C	DIO_SS0_CHANNEL_12	1
125	7D	DIO_SS0_CHANNEL_13	2
126	7E	DIO_SS0_CHANNEL_14	3
127	7F	DIO_SS0_CHANNEL_15	4

PCIe Expansion Board DIO Channels – VL-MPEe-A1/2

Channel Number	Hex Code	Channel Name	VL-MPEe-A1/2 J1 Pin
128	80	DIO_AX_CHANNEL_1	18
129	81	DIO_AX_CHANNEL_2	19
130	82	DIO_AX_CHANNEL_3	20

PCIe Expansion DIO Channels – VL-MPEe-U2

Channel Number	Hex Code	Channel Name	VL-MPEe-U2 J3 Pin
160	A0	DIO_U2_CHANNEL_0	1
161	A1	DIO_U2_CHANNEL_1	2
162	A2	DIO_U2_CHANNEL_2	3
163	A3	DIO_U2_CHANNEL_3	4
164	A4	DIO_U2_CHANNEL_4	6
165	A5	DIO_U2_CHANNEL_5	7
166	A6	DIO_U2_CHANNEL_6	8
167	A7	DIO_U2_CHANNEL_7	9
168	A8	DIO_U2_CHANNEL_8	11
169	A9	DIO_U2_CHANNEL_9	12
170	AA	DIO_U2_CHANNEL_10	13
171	AB	DIO_U2_CHANNEL_11	14
172	AC	DIO_U2_CHANNEL_12	15*

* This pin is Ground on standard products but can be made an I/O line on custom products.

Analog to Digital (ADC) Channel Assignments

The definitions below, which can be found in the VersaAPI header file, are organized by board type and channel number:

- CPU Board On-Board ADC Channels
- VL-SPX-1 Expansion Board ADC Channels by Slave Select
- PCIe Expansion Board ADC Channels for the VL-MPEe-A1/2

CPU Board On-board ADC Channels

Channel Number	Hex Code	Channel Name	EBX-37 J22 Pin	EBX-41 J19 Pin	EPIC-17 J18 Pin	EPIC-25 J25 Pin
0	00	SPI_AI_CHANNEL_1	1	1	1	1
1	40	SPI_AI_CHANNEL_2	2	2	2	2
2	10	SPI_AI_CHANNEL_3	3	3	3	3
3	50	SPI_AI_CHANNEL_4	4	4	4	4
4	20	SPI_AI_CHANNEL_5	6	6	6	6
5	60	SPI_AI_CHANNEL_6	7	7	7	7
6	30	SPI_AI_CHANNEL_7	8	8	8	8
7	70	SPI_AI_CHANNEL_8	9	9	9	9
8	01	SPI_AI_CHANNEL_9	—	11	11	—
9	41	SPI_AI_CHANNEL_10	—	12	12	—
10	11	SPI_AI_CHANNEL_11	—	13	13	—
11	51	SPI_AI_CHANNEL_12	—	14	14	—
12	21	SPI_AI_CHANNEL_13	—	16	16	—
13	61	SPI_AI_CHANNEL_14	—	17	17	—
14	31	SPI_AI_CHANNEL_15	—	18	18	—
15	71	SPI_AI_CHANNEL_16	—	19	19	—

VL-SPX-1 Expansion Board ADC Channels – Using Slave Select 0

Channel Number	Hex Code	Channel Name	VL-SPX-1 J2 Pin
64	04	AI_SS0_CHANNEL_1	1
65	44	AI_SS0_CHANNEL_2	2
66	14	AI_SS0_CHANNEL_3	3
67	54	AI_SS0_CHANNEL_4	4
			J3 Pin
68	24	AI_SS0_CHANNEL_5	1
69	64	AI_SS0_CHANNEL_6	2
70	34	AI_SS0_CHANNEL_7	3
71	74	AI_SS0_CHANNEL_8	4

VL-SPX-1 Expansion Board ADC Channels – Using Slave Select 1

Channel Number	Hex Code	Channel Name	VL-SPX-1 J2 Pin
80	05	AI_SS1_CHANNEL_1	1
81	45	AI_SS1_CHANNEL_2	2
82	15	AI_SS1_CHANNEL_3	3

83	55	AI_SS1_CHANNEL_4	4
			J3 Pin
84	25	AI_SS1_CHANNEL_5	1
85	65	AI_SS1_CHANNEL_6	2
86	35	AI_SS1_CHANNEL_7	3
87	75	AI_SS1_CHANNEL_8	4

VL-SPX-1 EXPANSION BOARD ADC CHANNELS – USING SLAVE SELECT 2

Channel Number	Hex Code	Channel Name	VL-SPX-1 J2 Pin
96	06	AI_SS2_CHANNEL_1	1
97	46	AI_SS2_CHANNEL_2	2
98	16	AI_SS2_CHANNEL_3	3
99	56	AI_SS2_CHANNEL_4	4
			J3 Pin
100	26	AI_SS2_CHANNEL_5	1
101	66	AI_SS2_CHANNEL_6	2
102	36	AI_SS2_CHANNEL_7	3
103	76	AI_SS2_CHANNEL_8	4

VL-SPX-1 EXPANSION BOARD ADC CHANNELS – USING SLAVE SELECT 3

Channel Number	Hex Code	Channel Name	VL-SPX-1 J2 Pin
112	07	AI_SS3_CHANNEL_1	1
113	47	AI_SS3_CHANNEL_2	2
114	17	AI_SS3_CHANNEL_3	3
115	57	AI_SS3_CHANNEL_4	4
			J3 Pin
116	27	AI_SS3_CHANNEL_5	1
117	67	AI_SS3_CHANNEL_6	2
118	37	AI_SS3_CHANNEL_7	3
119	77	AI_SS3_CHANNEL_8	4

PCIE EXPANSION BOARD ADC CHANNELS – VL-MPEE-A1/2

Channel Number	Hex Code	Channel Name	VL-MPEe-A1/2 J1 Pin
224	0E	PCI_AI_CHANNEL_1	1
225	4E	PCI_AI_CHANNEL_2	2
226	1E	PCI_AI_CHANNEL_3	5
227	5E	PCI_AI_CHANNEL_4	6
228	2E	PCI_AI_CHANNEL_5	9
229	6E	PCI_AI_CHANNEL_6	10
230	3E	PCI_AI_CHANNEL_7	13
231	7E	PCI_AI_CHANNEL_8	14

Digital to Analog (DAC) Channel Assignments

The definitions below are organized by board type and channel number:

- CPU Board On-Board DAC Channels
- VL-SPX-4 Expansion Board DAC Channels by Slave Select

CPU Board On-board DAC Channels

Channel Number	Hex Code	Channel Name	EBX-37 J22 Pin	EBX-41 J19 Pin	EPIC-17 J18 Pin	EPIC-25 J25 Pin
0	00	SPI_A0_CHANNEL_1	21	21	21	11
1	01	SPI_A0_CHANNEL_2	22	22	22	12
2	02	SPI_A0_CHANNEL_3	23	23	23	13
3	03	SPI_A0_CHANNEL_4	24	24	24	14
4	04	SPI_A0_CHANNEL_5	26	26	26	—
5	05	SPI_A0_CHANNEL_6	27	27	27	—
6	06	SPI_A0_CHANNEL_7	28	28	28	—
7	07	SPI_A0_CHANNEL_8	29	29	29	—

VL-SPX-4 Expansion Board DAC Channels – Using Slave Select 0

Channel Number	Hex Code	Channel Name	VL-SPX-4 J2 Pin
64	40	A0_SS0_CHANNEL_1	1
65	41	A0_SS0_CHANNEL_2	2
66	42	A0_SS0_CHANNEL_3	3
67	43	A0_SS0_CHANNEL_4	4

VL-SPX-4 Expansion Board DAC Channels – Using Slave Select 1

Channel Number	Hex Code	Channel Name	VL-SPX-4 J2 Pin
80	50	A0_SS1_CHANNEL_1	1
81	51	A0_SS1_CHANNEL_2	2
82	52	A0_SS1_CHANNEL_3	3
83	53	A0_SS1_CHANNEL_4	4

VL-SPX-4 Expansion Board DAC Channels – Using Slave Select 2

Channel Number	Hex Code	Channel Name	VL-SPX-4 J2 Pin
96	60	A0_SS2_CHANNEL_1	1
97	61	A0_SS2_CHANNEL_2	2
98	62	A0_SS2_CHANNEL_3	3
99	63	A0_SS2_CHANNEL_4	4

VL-SPX-4 EXPANSION BOARD DAC CHANNELS – USING SLAVE SELECT 3

Channel Number	Hex Code	Channel Name	VL-SPX-4 J2 Pin
112	70	A0_SS3_CHANNEL_1	1
113	71	A0_SS3_CHANNEL_2	2
114	72	A0_SS3_CHANNEL_3	3
115	73	A0_SS3_CHANNEL_4	4

Sample Code

B

```
/*                          VersaLogic VersaAPI Sample Code
 *
 *      This code is given with no guarantees. This code intended to be used as an example of how
 *      to use the VersaAPI software package and to check the operation post install. This code
 *      will only work with the VersaLogic VL-MPEe-A1/A2 miniPCIe board.
 *
 *      Program Description: This code will continuously toggle the output of channel 3 (pin 20 of
 *      the J1 connector on the VL-MPEe-A1/2) until the input of channel 2 (pin 19) goes low.
 *      This illustrates how the input from one channel can control the output of another channel.
 */

#include <VL_OSALib.h> // Include statement for the VersaAPI Header file.

#define sleep   50000   // This number is used to for defining the frequency of the output.

int main()
{
        // Program variable definitions.
        int n = 0;
        unsigned char loop = true;

        // VL_Open() statement must be called before using VersaAPI Channel calls.
        VL_Open();

        // User must set the channels to input or output before being able to use the DIO
        // Channel.
        VL_DIOSetChannelDirection( DIO_AX_CHANNEL_2, DIO_INPUT );
        VL_DIOSetChannelDirection( DIO_AX_CHANNEL_3, DIO_OUTPUT );

        // Loop statement to give the output a 50% duty cycle.
        while(loop)
        {
                n++;    // Counter post increment.

                // Simple 'if' statement to toggle an output.
                if (n == sleep)
                {
                        // Set the output HIGH.
                        VL_DIOSetChannelLevel( DIO_AX_CHANNEL_3, DIO_CHANNEL_HIGH );
                }
                else if (n == sleep + sleep)
                {
                        // Set the output LOW.
                        VL_DIOSetChannelLevel( DIO_AX_CHANNEL_3, DIO_CHANNEL_LOW );
                        n = 0;
                }

                // While this input returns a digital high the loop will continue
                // If the digital input signal goes low the loop will exit.
                loop = VL_DIOGetChannelLevel( DIO_AX_CHANNEL_2 );
        }

        // VL_Close must be called at the end of the program or when you no longer want to use
        // the DIO Channels.
        VL_Close();
        return(0);
}
```