Installation and Reference Guide

DOC. REV. 07/21/2021

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	Description
1.10	Update commercial release.
1.11	Added support for additional commands and boards
1.12	Added support for EPM-19. Updated **Error! Reference source not found.** through Table 23 with correct names and pin assignments.
1.14	Added support for additional commands and boards. Updated installation instructions for Linux release.
1.15	Added support for the EPU-3311. Added I2C and LCD backlight commands for the EPU-3311 and EPU-3312.
1.16	Added support for the EPU-3312. Fixed Channel Names in Table 13. PCIe Expansion DIO Channels – VL-MPEe-U2
1.17	Added support for the EPMe-42, EPM-43 and EBX-38.
1.18	Added support for the EPU-4x62 and EPU-4460.
1.19	Added support for generic FPGA read/write commands. Updated installation instructions for Windows release.
1.20	Added support for SPI bus configuration and read/write commands.
1.21	Added Digital I/O Channel Assignments for EPM-39.
1.22	Updated Windows 10 installation method.
1.23	Corrected DIO channel assignments for EPU-3312.
1.24	Added DIO channel assignments for EPU-401x.
1.25	Added procedure for installing the VL MPEe A1/A2 driver on Windows 10.
1.26	Added CAN API calls and documentation.
1.27	Added ESU-5070 timer calls.
1.28	API Release 1.6.1.
1.29	API Release 1.7.0. Support for EPMe-51. Updates to the MPEu-C1E sections. Added MPEu-C1E API calls VSL_CANOpenPortWithUserTiming(), VSL_CANGetProtocolRegisterStatus() and VSL_CANGetErrorCounterRegisterStatus(). Added additional LCD panel APIs.
1.30	API 1.7.1 Release Widows updates.
1.31	API 1.7.2 Release. Update to Linux installation section.

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.

Known Issues

Drivers/VersaAPI

- Driver support for VL-SPX-5 Solid State I/0 module and PWM output is not yet available.
- VersaAPI support for more than two (2) VL-SPX-X expansion modules connected simultaneously is not yet available.
- VersaAPI support for the three 8254 timers is not yet available on the VL-EBX-18 and VL-EPM-19.
- VersaLogic MPEe A1/A2 Driver is not supported on Windows 10.
- VersaLogic ARM boards only support DIO and Watchdog functions.
- VersaLogic does not officially support CentOS. The individual driver Makefile contains comments to assist with compiling and linking for CentOS. Search the VersaLogic VersaTech KnowledgeBase for CentOS compilation hints.
- VersaAPI does not support GPIO interrupts on the VL-MPEe-U2.
- User defined interrupts not supported in Windows OS.

Contents

Figures

Tables

Overview 1

Description

VersaAPI is a library of API function calls to help access various features of a VersaLogic board. VL_OSALib.h contains the API header information. To use the VersaAPI calls your application must include the header file VL_OSALib.h and link with the library file VL_OSALib.

Supported VersaLogic SBC Boards

Table 1 lists the VersaLogic Single Board Computer (SBC) boards and functions supported by VersaAPI (as of the publication date of this document).

All VersaLogic boards support the VersaAPI Administrative functions.

Table 1: Supported VersaLogic SBC Boards and Functions

Board	Function						
	ADC	**DAC**	**DIO**	**Counter Timer**	**Heat Sink Fan**	**I2C**	**LCD Panel**
Anaconda (VL-EBX-18)	✓	✓	✓	✓	—	—	—
Baycat (VL-EPM-31)	✓	✓	✓	✓	✓	—	—
Bengal (VL-EPMe-30)	✓	✓	✓	✓	✓	—	—
Blackbird (VL-EPU-4X52)	✓	✓	✓	✓	✓	✓	✓
Condor (VL-EPU-4460)	—	—	✓	✓	✓	✓	✓
Copperhead (VL-EBX-41)	✓	✓	✓	✓	✓	—	—
Fox (VL-EPM-19)	✓	✓	✓	✓	—	—	—
Grizzly (VL-ESU-5070)	—	—	✓	✓	✓	✓	—
Harrier (VL-EPU-4011)	—	—	✓	✓	—	—	—
Iguana (VL-EPIC-25)	✓	✓	✓	✓	✓	—	—
Komodo (VL-EPICs-36)	✓	✓	✓	—	—	—	—
Lion (VL-EPMe-42)	✓	✓	✓	✓	✓	—	—
Liger (VL-EPM-43)	✓	✓	✓	✓	✓	—	—
Mamba (VL-EBX-37)	✓	✓	✓	✓	—	—	—
Newt (VL-EPIC-17)	✓	✓	✓	✓	—	—	—
Osprey (VL-EPU-3311)	—	—	✓	✓	—	✓	✓
Owl (VL-EPU-4012)	✓	—	✓	✓	—	—	—
Raven (VL-EPU-3312)	✓	✓	✓	✓	✓	✓	✓
Sabertooth (VL-EPMe-51)	—	—	✓	✓	—	—	—
Sandcat (VL-EPM-39)	—	—	✓	✓	—	—	—
Viper (VL-EBX-38)	✓	✓	✓	✓	—	—	—

Supported VersaLogic Expansion Boards

Table 2 lists the VersaLogic boards and functions supported by VersaAPI (as of the publication date of this document).

VersaAPI supports each of these boards running on a VersaLogic SBC as well as running on a third-party board.

Table 2: Supported VersaLogic Expansion Boards and Functions

Board	Function		
	ADC	**DAC**	**DIO**
SPX-1	✓	—	—
SPX-2	—	—	✓
SPX-4	—	✓	—
VL-MPEe-A1/A2	✓	—	✓
VL-MPEe-U2	—	—	✓

Operating Systems

Various versions of VersaAPI will run under the following operating systems. Refer to the VersaLogic Software Support page for details, or to inquire regarding support for an OS not listed below:

- Windows 7 (32 bit)
- Windows 10 (32 bit)
- Windows 10 (64 bit)
- Ubuntu Linux (32 bit)
- Ubuntu Linux (64 bit)
- CentoOS Linux (64 bit) – Not officially supported, but driver Makefile have notes for compiling and linking.

Installation 2

Windows Installation

The following procedures are 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 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
- VL_OSALib.dll – Dynamic Link Library for the executable (EXE)
- VL_OSALib.h – Header file with all VersaAPI definitions
- VL_OSALib.lib – Library file for linking the DLL to the EXE when compiling
- VLDrive.inf – CPU board installation file
- VLDrive.sys – CPU board driver
- VLDrivep.inf – Mini PCIe module device driver installation file
- VLDrivep.sys – Mini PCIe module device driver
- wdfCoInstaller01009.dll - Microsoft Windows dll file

For the EPU products, the following library and driver are also included in the package:

- Cgos.sys – Carrier Group driver
- Cgos.cat - Carrier Group catalogue file
- Cgos.inf - Carrier Group installation file
- Cgos.dll – Dynamic Link Library for the executable (EXE)
- Cgos.lib – Library file for linking the DLL to the EXE when compiling
- Cgos.h – Carrier Group header file

INSTALLING THE DRIVERS

Note: Warning messages encountered when installing the drivers may be ignored.

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

Note: There is a special procedure for installing the VL-MPEe-A1/A2 driver on Windows 10. Please see instructions following this section.

1. Save the VersaLogic VersaAPI release package to a location accessible to the hardware.

2. Open the Windows Hardware Wizard.

   a. In Windows 7, using the Start menu search bar, type `hdwwiz`.
   b. In Windows XP, from the Control Panel, double-click the Add New Hardware icon.

   c. In Windows 10, as shown below, open the Device Manager then select "Add legacy hardware."

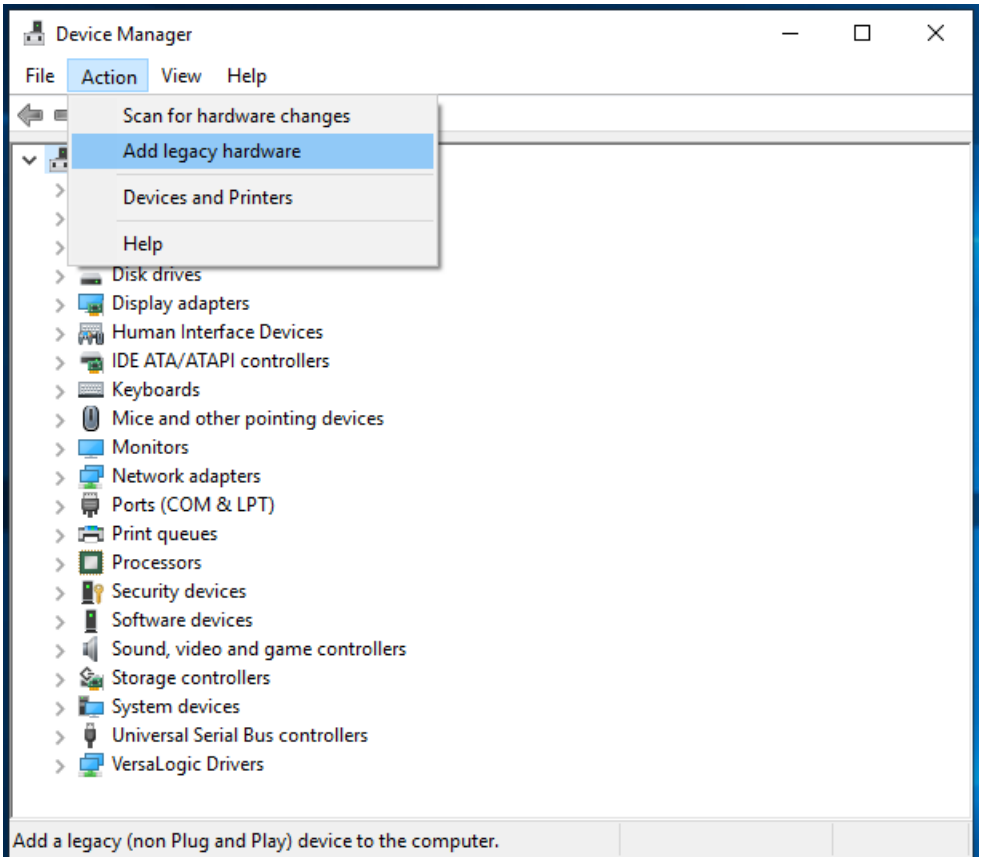

Figure 1. Add Legacy Hardware screen

3. Select the option to install the hardware manually and click Next. (Do not select the option for Windows to search for hardware.)

4. When the list of devices appears, select Show All Devices and click Next.

5. Click Have Disk.

6. Click Browse.

7. Navigate to the folder where you saved the VersaAPI package.

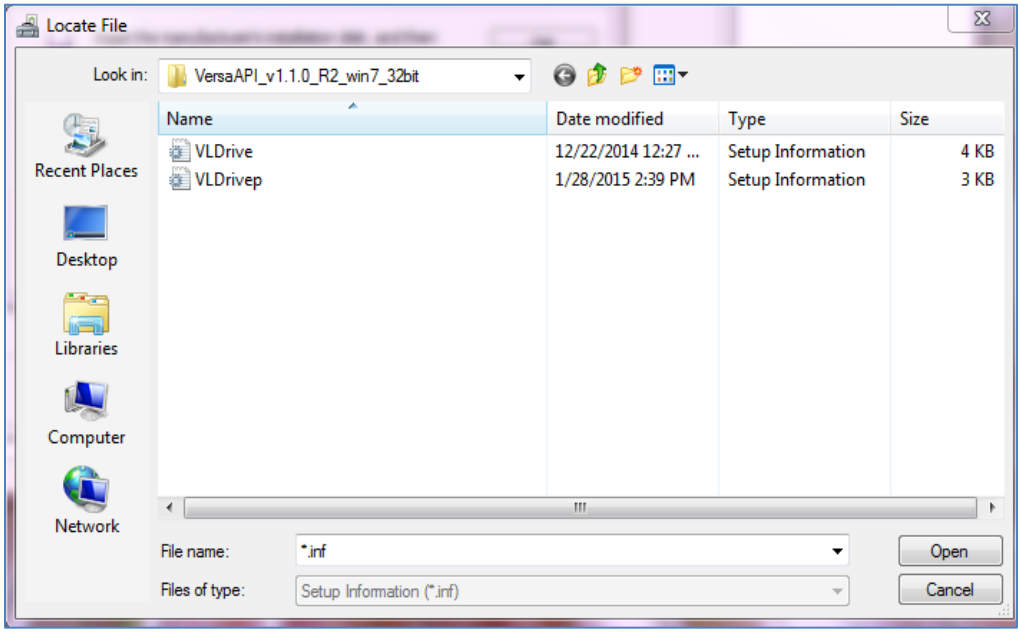

Figure 2. Selecting the VersaAPI Drivers

8. The setup information file (INF) you install depends on which VersaLogic boards and I/O functions you will be using:

    a. For VersaLogic CPU or SPX expansion boards select VLDrive.inf.
    b. For the VersaLogic VL-MPEe-A1/A2 module, select VLDrivep.inf.
    c. For the VersaLogic VL-EPU-33xx or VL-EPU-4x6x boards you will also install the Cgos.inf.

    Note: Install both drivers when using a CPU board and a Mini PCIe module.

9. Click Open.
10. Click OK. You are given the option to install one of three drivers:
    - VersaLogic Baseboard Driver
    - VersaLogic PCI Device Driver
    - VersaLogic Carrier Group Driver

11. Select the appropriate driver for the VersaLogic Embedded Computer or mini-PCIe module.
12. Complete the onscreen instructions.
13. Repeat the process for any other VersaAPI drivers needed (vldrive.inf, vldrivep.inf, or cgos.inf).
14. Set the VersaLogic driver I/O resource for your board (see Appendix B for details) and restart the computer when prompted.

INSTALLING THE VL-MPEE-A1/A2 DRIVER ON WINDOWS 10

1. Completely uninstall/remove any previously installed MPEe-A1/A2 driver from the Device Manager.
2. Reboot the computer and now you should see the "PCI Data Acquisition and Signal Processing Controller" listed under Other Devices.

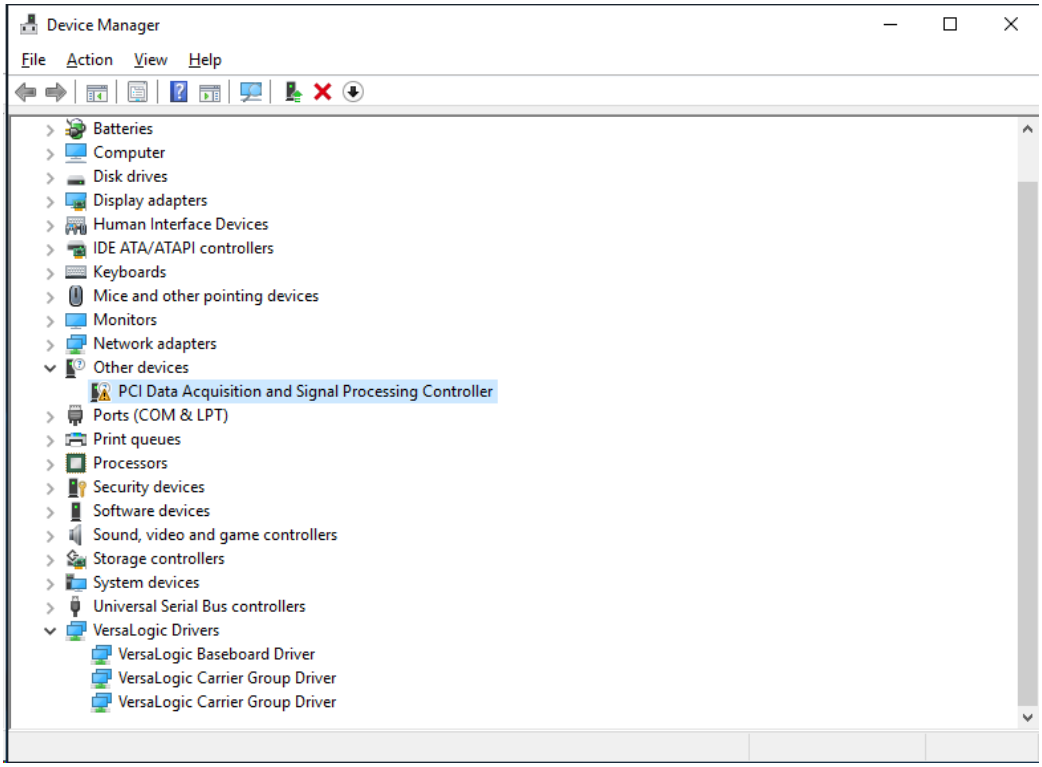

Figure 3. VL-MPEe-A1/A2 Driver screen

3. Right-click on "PCI Data Acquisition and Signal Processing Controller" and choose "Update Drivers".
4. Choose the second option "Browse my computer for driver software"
5. Under "Search for drivers in this location," type in or browse to the location where the driver is saved.
6. Click Next and the update tool should then find and install the MPEe-A1/A2 device driver.

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.

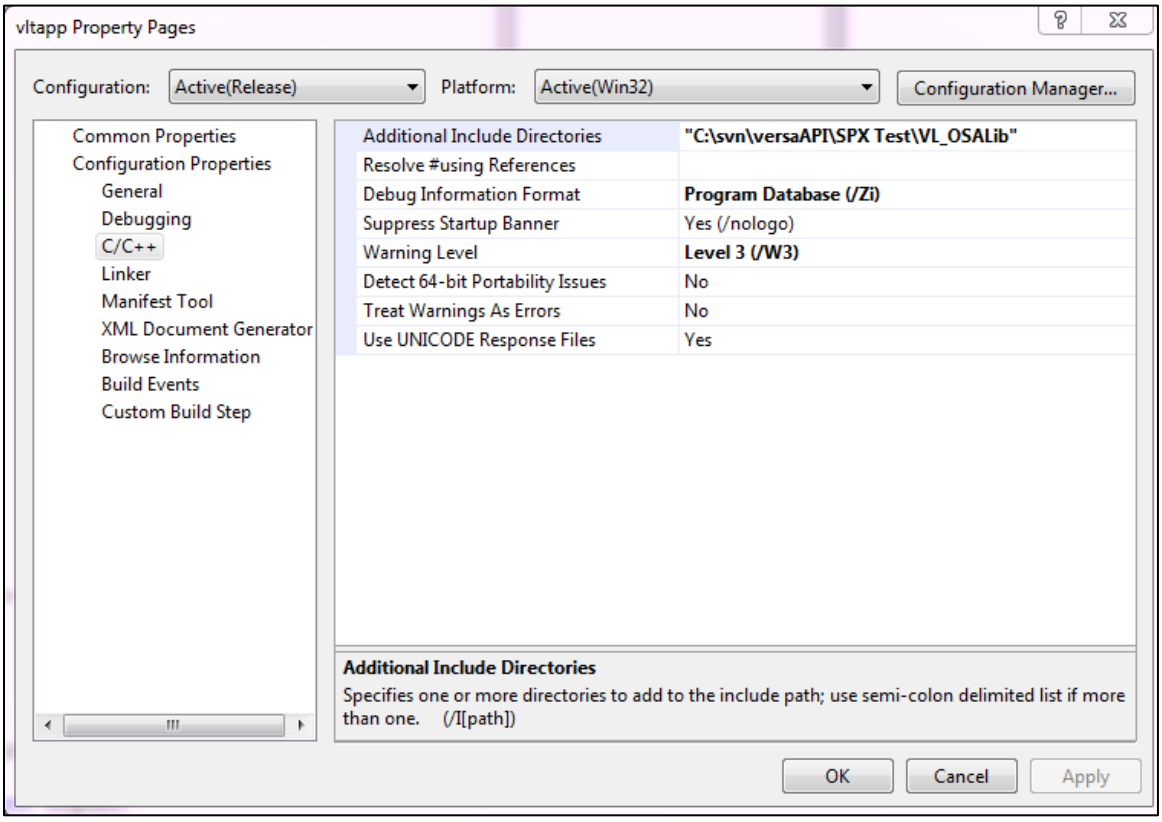

Figure 4. Including the Header File

6. Click OK. The header file has now been added to the project.
7. If building an x64 executable, then be sure to add "WIN32" to Preprocessor Definitions, so all the Windows code will be included in the build.

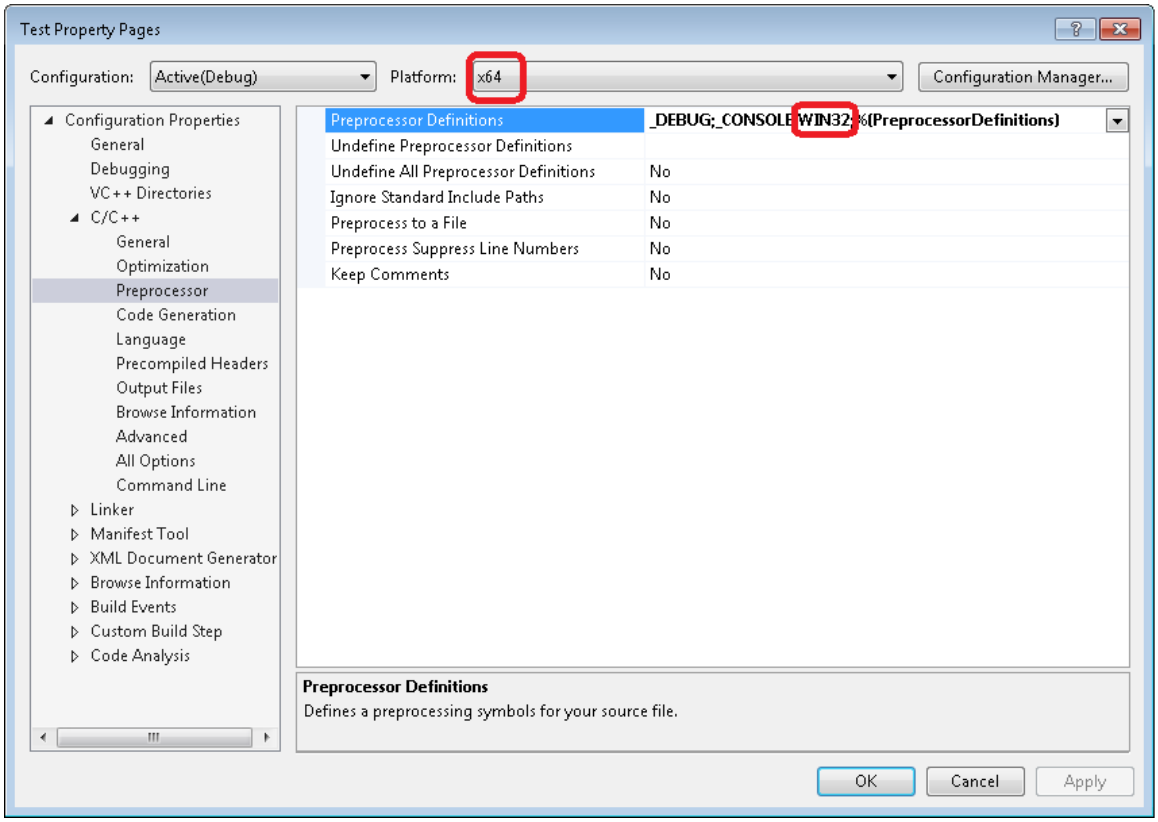

Figure 5. Visual Studio 2017 Preprocessor screen

LINKING THE LIBRARY

1. In Visual Studio, open the Properties for your project.
2. In Project Properties → Configuration Properties → Linker, select Input.
3. In Additional Dependencies, enter `vl_osalib.lib`.

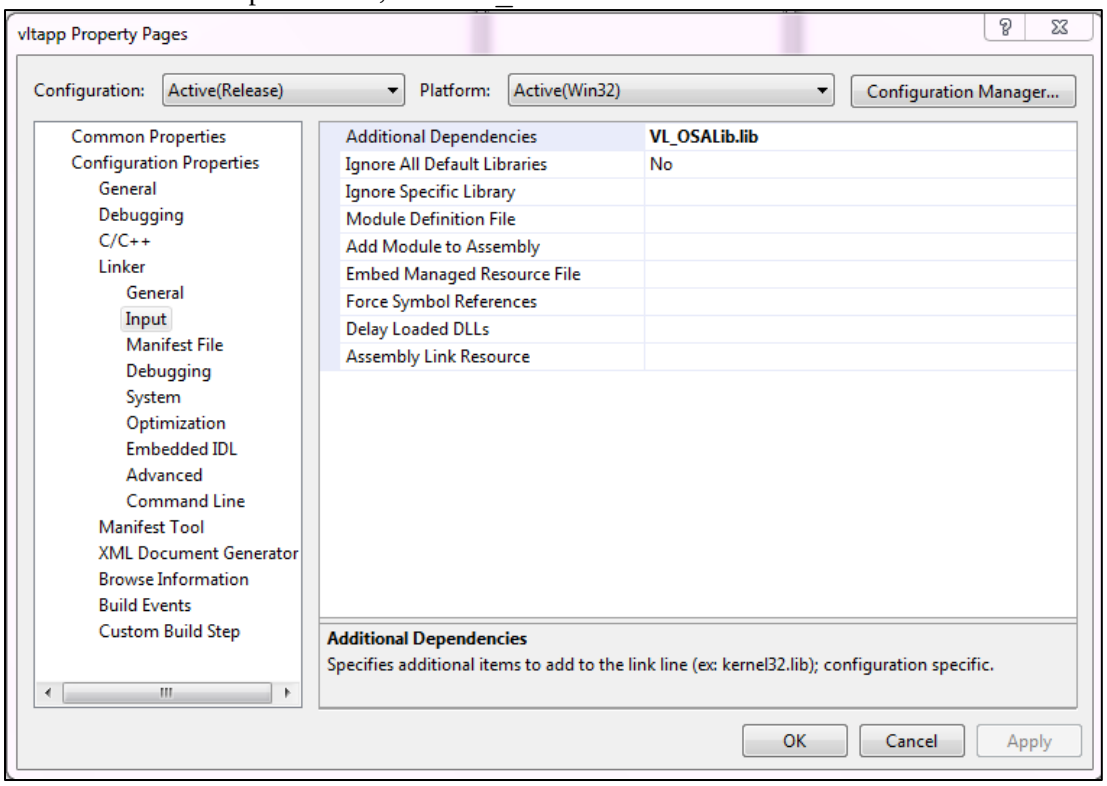

Figure 6. Linking the VersaAPI Library File

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.

6. If using a VersaLogic EPU product, repeat steps 1 -5 for the VersaLogic Carrier Group library Cgos.lib.

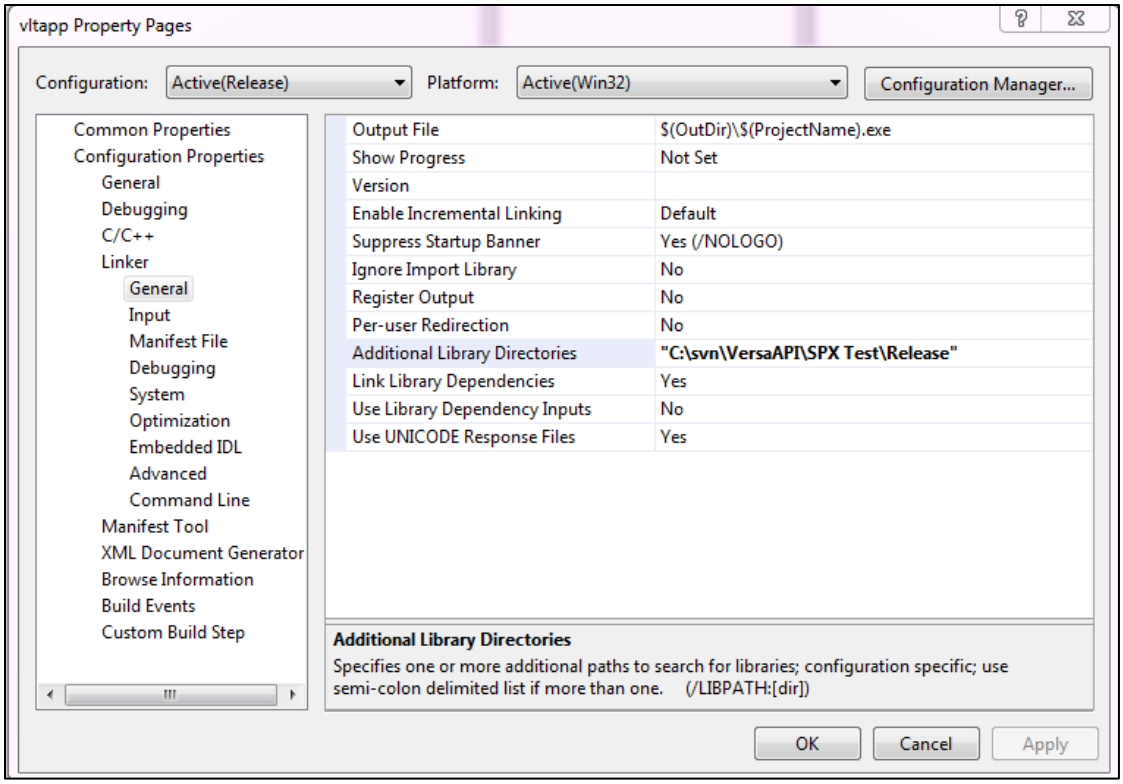

Figure 7. Adding the Library Folder

INCLUDING THE DLL IN THE PROJECT

The VL_OSALib.dll file (and Cgos.dll if using an EPU product) 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.

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.

Linux Installation

The following procedures are guidelines for installing VersaAPI on a machine running a Linux operating system. Some modifications to the procedures may be required depending on the version of Linux running. Contact VersaLogic Technical Support if you encounter problems with the installation.

SETUP

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

PACKAGE CONTENTS

Table 3: Package content for all boards

File Name	Function
vl_install.sh	VersaAPI install script
vl_install_can.sh	VersaAPI install script for the MPEu-C1E CAN modules
libVL_OSALib.so.X.Y.Z	Shared library file for linking with applications
VL_OSALib.h	Header file containing all necessary VersaAPI definitions
src	Directory of source code for the VersaLogic drivers
src/vldrive	CPU board device driver
src/vldrivep	Mini-PCIe module device driver. Needed for boards where the FPGA is located on the PCIe bus
src/vldriveax	MPEe-A1, MPEe-A2 module device driver
VersaAPIGuid_v1.7.0.pdf	VersaAPI Installation and Reference Guide
Examples	Directory containing example code to access various features on various VersaLogic boards. The example 'versaAPI_sample' contains example code for DIO, 8254 Timer, MPEe-Ax, Watchdog Timer and VL-SPXx functionality that is common across many of VersaLogic's SBCs.

For our EPU-331x products, the following library and driver are also included in the package:

Table 4: Package contents for EPU-331x boards

File Name	Function
src/vldrivecb	Carrier Group module device driver
libcgos.so	Additional shared library file for the Carrier Group boards
Cgos.h	Additional header file containing all necessary VersaAPI definitions for our Carrier Group boards

INSTALLING THE LIBRARY AND DRIVERS

To compile the VersaAPI Linux drivers, the appropriate Linux kernel header files are necessary.

Use the Linux installation script *vl_install.sh* to install the VersaAPI shared library into the /usr/local/lib directory, and to compile and install the necessary drivers into the /lib/modules/*kernel-version*/ directory:

1. Download and install the Linux kennel header files for your version of Linux.
2. From a Linux shell, run the supplied install script *vl_install.sh* using sudo:

   sudo ./vl_install.sh <*VersaLogic_Board_Name*>

   Where <*VersaLogic_Board_Name*> must be one of the following strings:

   - EBX-18
   - EPM-31
   - EPMe-39
   - EPU-3312
   - EPU-4460
   - MPEe-A1/A2
   - EPMe-5120
   - EBX-38
   - EPM-43
   - EPMe-42
   - EPU-4011
   - EPU-4x62
   - MPEe-U2
   - EPM-19
   - EPMe-30
   - EPU-3311
   - EPU-4012
   - ESU-5070
   - EPMe-51

2. Reboot the VersaLogic board.

After successful execution of *vl_install.sh* and a reboot of the board, you will be able to build applications and link the VersaAPI shared library by:

- Including VL_OSALib.h in your source file.
- Linking to the VersaLogic API libraries using "-lVL_OSALib"
- Linking to the VersaLogic Carrier Group API library "-lcgos" in your linker options if using an EPU-3311, EPU-3312, EPU-4x62 or EPU-4460.

Once successfully installed, delete the temporary directory created during Setup Step 2.

DRIVER COMMAND LINE OPTIONS

The CPU board device driver vldrive.ko has the following command line parameters:

> **IRQNum**: Informs the SBC FPGA which IRQ to use when interrupting the SBC. Default is IRQ 5.

To set the VersaLogic driver I/O resource manually, see Appendix B for details. After the resource is changed, restart (reboot) the computer for the changed option(s) to take effect.

MANUALLY COMPILING AND INSTALLING THE LIBRARY AND DRIVERS

Running script *vl_install.sh* will compiled and installed the VersaAPI library and drivers. It is also possible to compile and install the drivers manually. The "src" directory contains the driver source code. Install the Linux kernel header files prior to building the VersaLogic drivers. The "src" directory contains four subdirectories, one subdirectory for each driver. The vldrive directory contains the driver for VersaLogic CPU onboard I/O resources, used for most I/O functions (DIO, ADC, DAC, 8254 Timers, Watchdog Timers, etc.) on VersaLogic CPU boards. The vldriveax directory contains the driver for the VL-MPEe-A1/A2. The vldrivep directory contains the driver for the VL-MPEe-U2 accessory module and other PCI devices. The vldrivecb directory contains an additional driver (besides vldrive) for use on Carrier Group boards (the VL-EPU-3311, VL-EPU-3312, etc.). The steps to make and install each driver are fundamentally the same.

Manually Compiling vldrive Driver

When compiling the VersaLogic SBC driver vldrive manually, the FPGA_BASE variable needs to be set to the correct FPGA address. See Appendix B for the correct FPGA_BASE address to use on the command line below. This variable is set in the vldrive/Makefile file on line 6 (DRIVER_ARGS := "FPGA_BASE=0xC80").

In the driver directory there is a Makefile. To build and install the vldriveax and vldrivep driver, change to the relevant directory and issue the commands:

```
make clean; make build_<FPGA_BASE>; make install
```

Example for a 'Basic Configuration 0002' board:

```
make clean; make build_0xC80; make install
```

These commands will build the driver and install the driver into the file system so that on each boot the drivers will automatically start.

Manually Compiling vldriveax, vldrivep and vldrivecb Drivers

In each driver directory there is a Makefile. To build and install the vldriveax and vldrivep driver, change to the relevant directory and issue the commands:

```
make clean; make; make install
```

These commands will build the driver and install the driver into the file system so that on each boot the drivers will automatically start.

For assistance in manually compiling the VersaAPI drivers and applications, search the VersaLogic VersaTech KnowledgeBase.

VersaAPI Calls

3

General API Information

Every application that wishes to access VersaLogic board features by using the VersaAPI library interface must open the library prior to making any other API calls and must close the library before terminating the application.

Many API calls return a VL_StatusT type return code. For successful API calls, VL_API_OK is returned. For unsuccessful API calls, an appropriate return code for the error will be returned and any requested data will not be valid.

Table 5: API Return Codes (type VL_StatusT)

Return Code	Description
VL_API_OK	Command completed successfully
VL_API_ERROR	Unexpected error occurred
VL_API_INVALID_ARG	Command argument out of range
VL_API_NOT_SUPPORTED	Command not supported on this board
VL_API_ARG_NOT_SUPPORTED	Argument currently not supported
VL_API_BC_LIBRARY_INIT_ERROR	API library initialization error
VL_API_BC_BOARD_OPEN_ERROR	Error opening Carrier Group error
VL_API_BC_LIBRARY_INSTALL_ERROR	Error instantiating Carrier Group library
VL_API_BC_LIBRARY_CLOSE_ERROR	Error closing Library
VL_API_I2C_REQUESTED_BUS_NOT_FOUND	Could not find the requested I2C bus
VL_API_I2C_READ_ERROR	Error attempting to read I2C address
VL_API_I2C_READ_REGISTER_ERROR	Error attempting to read I2C register
VL_API_I2C_WRITE_ERROR	Error attempting to write I2C address
VL_API_I2C_WRITE_REGISTER_ERROR	Error attempting to write I2C register
VL_API_GET_UPTIME_ERROR	Error attempting to get the board up time
VL_API_VGA_BL_VALUE_OUT_OF_RANGE	Specified backlight value is out of range
VL_API_VGA_BL_INTERFACE_ERROR	Unexpected backlight error
VL_API_IRQ_NUM_UNKNOWN	Could not determine current IRQ number
VL_API_BOARD_ID_ERROR	Error identifying board
VL_API_INVALID_DIO_CHANNEL	Invalid DIO Channel ID specified.
VL_API_SPI_READ_ERROR	Error attempting to read SPI data registers
VL_API_SPI_WRITE_ERROR	Error attempting to write SPI data registers
VL_API_CAN_CLOSE_DEVICE_ERROR	Could not close the CAN device
VL_API_CAN_OPEN_DEVICE_ERROR	Could not open the CAN device
VL_API_CAN_OPEN_NO_DEVICE_FOUND	Could not find any USB (CAN) device
VL_API_CAN_OPEN_NO_DEVICE_DESC	Could not find the CAN device descriptor
VL_API_CAN_OPEN_PORT_ERROR	Could not open the CAN device descriptor
VL_API_CAN_PORT_CLOSED	Could not close/port already closed CAN port
VL_API_CAN_TX_ERROR	Failed to transmit CAN packet

Administration Calls

API calls related to the library or board itself.

VSL_Open

Opens the VersaAPI library.

Syntax: VL_OSALIB_API unsigned long VSL_Open();

Inputs: None

Outputs: unsigned long
Returns 0 upon success and nonzero if no useable drivers are found.

VSL_Close

Closes the VersaAPI library.

Syntax: VL_OSALIB_API unsigned long VSL_Close();

Inputs: None

Outputs: Returns 0 upon success and nonzero if the drivers cannot close the library.

VSL_GetVersion

Retrieves the version number of the VersaAPI library

Syntax: VL_OSALIB_API void VSL_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.

VSL_GetProductInfo

Retrieves various product information for the board(s)

Syntax: VL_OSALIB_API void VSL_GetProductInfo();

Inputs: unsigned long ProductList
Should be 0x1.

unsigned char *BoardName
Pointer to a char array that will contain the board name.

unsinged char *Attributes
Pointer to a char

Outputs: None.
Product information will be displayed on the screen and written to a log file. Information includes board(s) product name; number of supported DIOs, timers and Watchdog timers; extended temperature support; FPGA revision level of the board.

VSL_GetUptime

Retrieves the current running time (uptime) of the board measured in hours.

Syntax: VL_OSALIB_API VSL_APIStatusT VSL_GetUptime(unsigned long *Uptime);

Inputs: unsigned long *Uptime
A pointer to the unsigned long where the value of the uptime will be stored.

Outputs: Returns VL_API_OK on successful retrieval of the uptime, otherwise returns VL_API_GET_UPTIME_ERROR.

Heat Sink Fan Calls

API calls to control or interrogate the heat sink fan.

VSL_FanOn

Turn the heat sink fan on.

Syntax: VL_OSALIB_API unsigned char VSL_FanOn();

Inputs: None.

Outputs: None.

VSL_FanOff

Turn the heat sink fan off.

Syntax: VL_OSALIB_API unsigned char VSL_FanOff();

Inputs: None.

Outputs: None.

VSL_FanGetRPM

Get the heat sink fan RPM sensor reading.

Syntax: VL_OSALIB_API unsigned char VSL_FanGetRPM(unsigned char PulsePerRevolution);

Inputs: unsigned char PulsePerRevolution
The number of pulses per revolution for your fan (usually specified in the fan data sheet). Typical value is 2.

Outputs: Revolutions per minute for the heat sink fan.

Digital I/O (DIO) Calls

API calls exist to control or interrogate DIO (also known as GPIO) channels.

DIO calls require channel identification. See Digital I/O (DIO) Channel Assignments for channel definitions.

Table 6 lists the level and direction parameter definitions used in DIO calls.

Table 6: DIO API Parameter Definitions

Parameters	Value	
Level	DIO_CHANNEL_LOW	0x00
	DIO_CHANNEL_HIGH	0x01
	DIO_CHANNEL_UNKNOWN	0x02
Direction	DIO_OUTPUT	0x00
	DIO_INPUT	0x01
	DIO_UNKNOWN	0.02
Interrupt Mode	DIO_INTERRUPT_ENABLED	0x01
	DIO_INTERRUPT_DISABLED	0x00
Interrupt Polarity	DIO_INTERRUPT_INVERT	0x01
	DIO_INTERRUPT_NOINVERT	0x00
Interrupt Level	DIO_INTERRUPT_HIGH	0x00
	DIO_INTERRUPT_LOW	0x01
	DIO_INTERRUPT_UNKNOWN	0x02
Interrupt Number	DIO_INTERRUPT_IRQ3	0x000
	DIO_INTERRUPT_IRQ4	0x001
	DIO_INTERRUPT_IRQ5	0x010
	DIO_INTERRUPT_IRQ10	0x011
	DIO_INTERRUPT_IRQ6	0x100
	DIO_INTERRUPT_IRQ7	0x101
	DIO_INTERRUPT_IRQ9	0x110
	DIO_INTERRUPT_IRQ11	0x111

VSL_DIOGetChannelLevel

Get the signal level of the specified channel.

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

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

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

VSL_DIOSetChannelLevel

Set the signal level of the specified channel.

Syntax: VL_OSALIB_API void VSL_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.

VSL_DIOSetChannelDirection

Set the signal direction of the specified channel.

Syntax: VL_OSALIB_API VSL_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.

VSL_DIOGetChannelDirection

Get the signal direction of the specified channel.

Syntax: VL_OSALIB_API VSL_DIOGetChannelDirection(unsigned char Channel, unsigned char *Direction);

Inputs: unsigned char Channel
The DIO channel number.

unsigned char Direction
Pointer to a buffer containing the current DIO direction: DIO_INPUT or DIO_OUTPUT.

Outputs: Returns VL_API_OK on success or
VL_API_INVALID_ARG if function argument is invalid.

VSL_DIOGetChannelInterruptEvent

Register an event handler with the Operating System. This event handler will be called by the operating system when the interrupt transitions.

Windows Operating System

Returns an unsigned integer that is actually a handle to an object that can be used in a wait type function (WaitForSingleObject() API call) in Windows. The object will be signaled when the interrupt transitions.

Syntax: VL_OSALIB_API unsigned int
VSL_DIOGetChannelInterruptEvent(unsigned char Timer)

Linux Operating System

If successful, will return the process id (pid) of the calling application, otherwise the hexadecimal value 0xff.

Syntax: VL_OSALIB_API unsigned int
VSL_DIOGetChannelInterruptEvent(unsigned char DIO_Channel, pid_t Pid)

Inputs: unsigned char Timer
One of the four DIO Channels .

pid_t Pid
The process id of the application for VersaAPI to signal when the interrupt transitions.

Outputs: unsigned int
Pid of the calling application (second argument to the function call).
0xff on error.

The user must provide a signal handler that the VersaAPI driver will signal using an interrupt after the interrupt transitions. The argument for the handler will be an unsigned integer that is a mask for the interrupt number.

Signal handler Syntax:

Syntax: void Signal_Handler_Function(int Signal, siginfo_t Info, void *Nothing)

The timer that generated the signal can be identified by the 'si_int' field in the Info structure (Info->si.int).

The signal handler must be registered with the application using the standard Linux command 'sigaction'.

To determine which timer caused the interrupt, check the **Info->si_int** field in the signal handler. The content of Info→si_int is defined as a bit mask as shown in table "DIO API Parameter Definitions" under "Interrupt Number":

Analog-to-Digital Conversion (ADC) Calls

These API calls 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.

Table 7 lists the range and format parameter definitions used in ADC calls.

Table 7: ADC API Parameter Definitions

Parameter	Value	
Range	SPI_RANGE_PM5V	0x00
	SPI_RANGE_PM10V	0x04
	SPI_RANGE_0_5V	0x08
	SPI_RANGE_0_10V	0x0C
Format	AI_RAW	0x00
	AI_VOLTS	0x01

VSL_ADCSetAnalogInputRange

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

Syntax: VL_OSALIB_API void VSL_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: None.

VSL_ADCGetAnalogInput

Get the value of the selected analog input channel.

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

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

unsigned char Format
Format of the returned data.

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.

VSL_GetADCType

Get the model of Analog Input card installed.

Syntax: VL_OSALIB_API unsigned char VSL_GetADCType();

Inputs: None.

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

These API calls 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.

VSL_DACSetAnalogOutput

Set the analog output channel and range for a selected channel.

Syntax: VL_OSALIB_API void VSL_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 emulate 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.

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

Table 8: 8254 Timer Parameters

	Parameter	Description
Name	VL_TIMER0	Timer 0
	VL_TIMER1	Timer 1
	VL_TIMER2	Timer 2
Mode	VL_TIMER_MODE0	Interrupt on terminal count
	VL_TIMER_MODE1	Hardware retriggerable one-shot
	VL_TIMER_MODE2	Rate generator
	VL_TIMER_MODE3	Square wave mode
	VL_TIMER_MODE4	Software triggered strobe
	VL_TIMER_MODE5	Hardware triggered strobe
Type	VL_TIMER_TYPE_INTERNAL	Internal timer type
	VL_TIMER_TYPE_EXTERNAL	External timer type

VSL_TMRSet

Start one of three 8254 timers. You must select a mode and duration.

Syntax: VL_OSALIB_API char VSL_TMRSet(unsigned char Timer, unsigned char Mode, unsigned short Count, BOOL Type);

Inputs:

unsigned char Timer
One of the three timers. Table 8: 8254 Timer Parameters lists the valid Timer Name values.

unsigned char Mode
The 8254 timer mode to set. Table 8: 8254 Timer Parameters lists the valid Mode values.

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

unsigned char Type

Type of the timer. Must be either VL_TIMER_TYPE_EXTERNAL, or VL_TIMER_TYPE_INTERNAL (recommended).

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

VSL_TMRGetSettings

Get the settings for one of three 8254 timers.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_TMRGetSettings(unsigned char Timer, unsigned short Count, unsigned char *pMode, unsigned char *pType, unsigned char *pIntStatus, unsigned char *pIRQStatus, **unsigned char *pIRQEnabled, unsigned char *pIRQStatus, unsigned char *pClockSelect)**;

Inputs: unsigned char Timer
One of the three timers. Valid values are listed in table 5.

Outputs: unsigned short *pCount
Count value for the timer.

unsigned char *pMode
Timer mode value. Valid values are listed are listed in table 6.

unsigned char *pType
Type of counter. Valid values are listed in table 7.

unsigned char *pIntStatus
Interrupt status. Valid values are 1 (interrupt generation is enabled) and 0 (interrupt generation is disabled).

unsigned char *pIRQNum
IRQ number to use.

unsigned char *pIRQEnabled
Whether the timer will generate an IRQ when triggered. Valid values are 0 (enabled – IRQ will be generated) and 0 (disabled – IRQ will not be generated).

unsigned char *pIRQStatus
Status of the IRQ (if IRQ generation is enabled). Valid values are 1 (timer output has transitions from 0 to 1) and 0 (timer output has not transitioned from 0 to 1).

unsigned char *pClockSelect
Clock for the timer. Valid values are 0 (internal clock) and 1 (external clock)

VSL_TMRGet

Get the current count of the timer.

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

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

Outputs: unsigned short
The current count of the specified timer.

VSL_TMRClear

Stop the internal timer from counting down.

Syntax: VL_OSALIB_API void VSL_TMRClear(unsigned char Timer);

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

Outputs: None.

VSL_TMRGetEvent

Register an event handler with the Operating System. This event handler will be called by the operating system when the timer expires.

Windows Operating System

Returns an unsigned integer that is actually a handle to an object that can be used in a wait type function (WaitForSingleObject() API call) in Windows. The object will be signaled when the timer has expired.

Syntax: VL_OSALIB_API unsigned int VSL_TMRGetEvent(unsigned char Timer)

Linux Operating System

If successful, will return the process id (pid) of the calling application, otherwise the hexadecimal value 0xff.

Syntax: VL_OSALIB_API unsigned int VSL_TMRGetEvent(unsigned char Timer, pid_t Pid)

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

pid_t Pid
The process id of the application for VersaAPI to signal when the timer expires.

Outputs: unsigned int
Pid of the calling application (second argument to the function call).
0xff on error.

The user must provide a signal handler that the VersaAPI driver will signal using an interrupt after the timer has expired. The argument for the handler will be an unsigned integer that is a mask for the three timers.

Signal handler Syntax:

Syntax: void Signal_Handler_Function(int Signal, siginfo_t Info, void *Nothing)

The timer that generated the signal can be identified by the 'si_int' field in the Info structure (Info->si.int).

The signal handler must be registered with the application using the standard Linux command 'sigaction'.

To determine which timer caused the interrupt, check the **Info->si_int** field in the signal handler. The content of Info→si_int is 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

Counter/Timer (CTimer) Calls (ESU-5070 Grizzly only)

Each Counter/Timer is designed to count cycles of the internal clock and can optionally generate interrupts at timer values based on the specified match value.

VersaAPI simplifies access to the timers into six API calls.

Table 9: CTimer Names

Timer Name	Corresponding Timer
VL_TIMER0	Timer 0
VL_TIMER1	Timer 1

VSL_CTimerSet

Start the timer counting down.

Syntax: VL_OSALIB_API void VSL_CTimerSet(unsigned char Timer, unsigned char ResetEnable, unsigned char StopEnable, unsigned long MatchValue, unsigned char OutControl, unsigned char OutInitState, unsigned char IntEnable);

Inputs: unsigned char Timer
One of the two counter/timer names (VL_TIMER0|VL_TIMER1).

unsigned char ResetEnable

Enable/Disable reset of counter/timer when specified value is reached. 0 (default) do not reset the counter/timer; 1 reset the counter/timer.

unsigned char StopEnable

Stop/Do not stop the counter/timer when the value is reached. 0 (default) do not stop the counter/timer; 1 stop the counter/timer.

unsigned long MatchValue

Value to count to/match.

unsigned char OutControl

Action to take when counter/timer matches MatchValue. 0 (default) No action is taken; 1 Clear the output to 0; 2 Set output to 1; 3 Toggle the output.

unsigned char OutInitState

Initial state of the output.

unsigned char IntEnable

Enable/Disable interrupt on match. 0 (default) Do not generate interrupt on match; 1 Generate interrupt on match.

Outputs: None.

VSL_CTimerGet

Get the current count of the timer.

Syntax: VL_OSALIB_API unsigned long VSL_CTimerGet(unsigned char Timer);

Inputs: unsigned char Timer
One of the two counter/timer names (VL_TIMER0|VL_TIMER1).

Outputs: unsigned long
Current count of the specified counter/timer.

VSL_CTimerClear

Stop the internal timer from counting down.

Syntax: VL_OSALIB_API void VSL_CTimerClear(unsigned char Timer);

Inputs: unsigned char Timer
One of the two counter/timer names (VL_TIMER0|VL_TIMER1).

Outputs: None.

VSL_CTimerStart

Start the timer.

Syntax: VL_OSALIB_API void VSL_CTimerStart(unsigned char Timer);

Inputs: unsigned char Timer
One of the two counter/timer names (VL_TIMER0|VL_TIMER1).

Outputs: None.

VSL_CTimerStop

Stop the internal timer.

Syntax: VL_OSALIB_API void VSL_CTimerStop(unsigned char Timer);

Inputs: unsigned char Timer
One of the two counter/timer names (VL_TIMER0|VL_TIMER1).

Outputs: None.

VSL_CTimerReset

Stop the internal timer from counting down.

Syntax: VL_OSALIB_API void VSL_CTimerReset(unsigned char Timer);

Inputs: unsigned char Timer
One of the two counter/timer names (VL_TIMER0|VL_TIMER1).

Outputs: None

Watchdog Timer Calls

Not supported for the ESU-5070.

The Watchdog Timer has a 1-255 second programmable timeout. The Watchdog Timer, when triggered, can set a status bit, or execute a push-button-reset command. The timer starts to count down when enabled. The watchdog action triggers when the counter reaches zero (so a non-zero value should be set before enabling the Watchdog Timer). Software should periodically set a non-zero value for the Watchdog Timer to prevent triggering. The value written should always be one greater than the desired timeout value due to a 0-1 second error band. When triggered, the Watchdog Timer status will remain non-zero until one of the following happens:

- The Watchdog Timer is "petted" (reset) using one of the following VersaAPI commands:
  - A VSL_WDTSet()
  - A VSL_WDTEnable()
  - A VSL_WDTResetEnable()
- The board is reset.

VSL_WDTStatus

Retrieve whether the Watchdog Timer has triggered. This function will return the same value if it is enabled, or is disabled but has not triggered.

Syntax: VL_OSALIB_API unsigned char VSL_WDTStatus();

Inputs: None.

Outputs: unsigned char
Returns whether the Watchdog Timer has triggered. A return value of 0 implies that the Watchdog Timer is either disabled, or has not triggered. A return value of non-zero implies that the Watchdog Timer has triggered.

VSL_WDTEnable

Start (enable)/Stop (disable) the Watchdog Timer. If the Watchdog Timer is enabled, this call causes the Watchdog Timer to start counting from the last set value.

Syntax: VL_OSALIB_API void VSL_WDTEnable(unsigned char State);

Inputs: unsigned char State

ENABLE | DISABLE

Use ENABLE to start (enable) the Watchdog Timer and DISABLE to stop (disable) the Watchdog Timer.

Outputs: None.

VSL_WDTResetEnable

Set whether to perform a push-button reset command when the watchdog triggers. If the Watchdog Timer is enabled, this call causes the Watchdog Timer to start counting from the last set value.

Syntax: VL_OSALIB_API void VSL_WDTResetEnable(unsigned char State);

Inputs: unsigned char State

ENABLE | DISABLE

Use ENABLE to force (enable) a push-button reset when the watchdog triggers and DISABLE to stop (disable) a push-button reset when the watchdog triggers.

Outputs: None.

VSL_WDTSetValue

Set the number of seconds the Watchdog Timer will trigger once enabled.

Syntax: VL_OSALIB_API void VSL_WDTStatus(unsigned char Value);

Inputs: unsigned char Value

Number of seconds (between 0 and 255) to start counting down before the Watchdog Timer should trigger.

Outputs: None.

BIOS Calls

Not supported for the ESU-5070.

These API calls are used to query and set the BIOS bank to boot from on the next board power cycle/reset.

VSL_SetActiveBIOSBank

Set the active BIOS bank to either primary, or secondary (backup).

Syntax: VL_OSALIB_API VL_APIStatusT VSL_SetActiveBIOSBank(unsigned char bank);

Inputs: unsigned char bank
BIOS bank to used next power cycle/reset. Value must be either BIOS_PRIMARY_BANK or BIOS_SECONDARY_BANK.

Outputs: Returns VL_API_OK on success.
VL_API_INVALID_ARG if the input is invalid, or VL_API _ERROR for any other error.

VSL_GetActiveBIOSBank

Retrieve the active BIOS bank selection.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_GetActiveBIOSBank(unsigned char *bank);

Inputs: unsigned char *bank
Pointer to an unsigned char that will contain the current BIOS bank selection. Value will be BIOS_PRIMARY_BANK or BIOS_SECONDARY_BANK.

Outputs: Returns VL_API_OK.

I2C Calls

These API calls are used to access and control devices on the onboard I2C bus.

Note: Improper use of these functions could possibly lead to conditions preventing your system from booting. Use these commands with caution.

I2C calls require you to identify the specific bus you are accessing. Currently the only I^2C bus supported is the primary bus.

Table 10: I2C API Parameter Definitions lists the range and format parameter definitions used in I2C calls.

Table 10: I2C API Parameter Definitions

Parameters	Value	
Bus to access	VL_I2C_BUS_TYPE_PRIMARY	
Frequency	VL_I2C_FREQUENCY_100KHZ	100000
	VL_I2C_FREQUENCY_400KHZ	400000

VSL_I2CIsAvailable

Determine if I2C bus is present.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_I2CIsAvailable(unsigned long BusType);

Inputs: unsigned long BusType
Bus being accessed. Currently the only supported value is VL_I2C_BUS_TYPE_PRIMARY.

Outputs: Returns VL_API_OK on success or VL_API_I2C_REQUESTED_BUS_NOT_FOUND on error.

VSL_I2CReadAddress

Read NumSequentialBytes bytes of data from the specified address.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_I2CReadAddress(unsigned long BusType, unsigned char Address, unsigned char *pData, unsigned long NumSequentialBytes);

Inputs: unsigned long BusType
Bus being accessed. Currently the only supported value is VL_I2C_BUS_TYPE_PRIMARY.

unsigned char Address
8-bit address of the device to read on the bus. Bit 0 must be logical 1 to indicate a read operation.

unsigned char *pData
Pointer to the destination buffer

unsigned long NumSequentialBytes
Number of the sequential bytes to read.

Outputs: Returns VL_API_OK on success.
VL_API_I2C_REQUESTED_BUS_NOT_FOUND if the requested bus is not found or VL_API_I2C_READ_ERROR if an error occurred while attempting to read the data.

VSL_I2CReadRegister

Read one byte of data from the specified register and address.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_I2CReadRegister(unsigned long BusType, unsigned char Address, unsigned short RegisterNumber, unsigned char *pData);

Inputs: unsigned long BusType
Bus being accessed. Currently, the only supported value is VL_I2C_BUS_TYPE_PRIMARY.

unsigned char Address
8-bit address of the device to read on the bus. Bit 0 must be logical 1 to indicate a read operation.

unsigned short RegisterNumber
Number of the register to read.

unsigned char *pData
Pointer to the destination buffer

Outputs: Returns VL_API_OK on success.
VL_API_I2C_REQUESTED_BUS_NOT_FOUND if the requested bus is not found, or VL_API_I2C_READ_REGISTER_ERROR if an error occurred while attempting to read the register.

VSL_I2CWriteAddress

Write NumSequentialBytes bytes of data into the specified address.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_I2CWriteAddress(unsigned long BusType, unsigned char Address, unsigned char *pData, unsigned long NumSequentialBytes);

Inputs: unsigned long BusType
Bus being accessed. Currently, the only supported value is VL_I2C_BUS_TYPE_PRIMARY.

unsigned char Address
8-bit address of the device to write to on the bus. Bit 0 must be logical 0 to indicate a write operation.

unsigned char *pData
Pointer to the data source buffer.

unsigned long NumSequentialBytes
Number of the sequential bytes to write.

Outputs: Returns VL_API_OK on success.
VL_API_I2C_REQUESTED_BUS_NOT_FOUND if the requested bus is not found, or
VL_API_I2C_WRITE_ERROR if an error occurred while attempting to write the data.

VSL_I2CWriteReadCombined

Write NumSequentialBytes bytes of data into the specified address followed immediately by a read (no STOP condition send after the write, read is initiated with a start condition).

Syntax: VL_OSALIB_API VL_APIStatusT VSL_I2CWriteReadCombined(unsigned long BusType, unsigned char Address, unsigned char *pWriteData, unsigned long NumWriteBytes, unsigned char *pReadData, unsigned char *pNumReadBytes);

Inputs: unsigned long BusType
Bus being accessed. Currently, the only supported value is VL_I2C_BUS_TYPE_PRIMARY.

unsigned char Address
8-bit address of the device to write to on the bus. Bit 0 must be logical 0 to indicate a write operation.

unsigned char *pWriteData
Pointer to the data source buffer.

unsigned long NumWriteBytes
Number of the sequential bytes to write.

unsigned char *pReadData
Pointer to the data read buffer.

unsigned long NumReadBytes
Number of the sequential bytes read.

Outputs: Returns VL_API_OK on success.
VL_API_I2C_REQUESTED_BUS_NOT_FOUND if the requested bus is not found, or
VL_API_I2C_WRITE_ERROR if an error occurred while attempting to write the data.

VSL_I2CWriteRegister

Write one byte of data to the specified register and address.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_I2CWriteRegister(unsigned long BusType, unsigned char Address, unsigned short RegisterNumber, unsigned char *pData);

Inputs: unsigned long BusType
Bus being accessed. Currently, the only supported value is VL_I2C_BUS_TYPE_PRIMARY.

unsigned char Address
8-bit address of the device to write on the bus.Bit 0 must be logical 0 to indicate a write operation.

unsigned short RegisterNumber
Number of the register to write into.

unsigned char *pData
Pointer to the data source buffer.

Outputs: Returns VL_API_OK on success.
VL_API_I2C_REQUESTED_BUS_NOT_FOUND if the requested bus is not found, or VL_API_I2C_WRITE_REGISTER_ERROR if an error occurred while attempting to write into the register.

VSL_I2CGetMaxFrequency

Retrieve the maximum frequency of the specified I^2C bus.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_I2CGetMaxFrequency(unsigned long BusType, unsigned long *pMaxFrequency);

Inputs: unsigned long BusType
Bus being accessed. Currently, the only supported value is VL_I2C_BUS_TYPE_PRIMARY.

unsigned long *pMaxFrequency
Pointer to the data buffer.

Outputs: Returns VL_API_OK on success.
VL_API_I2C_REQUESTED_BUS_NOT_FOUND if the requested bus is not found, or VL_API_I2C_READ_REGISTER_ERROR if an error occurred while attempting to retrieve the maximum frequency of the bus.

VSL_I2CGetFrequency

Retrieve the current frequency of the specified I^2C bus.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_I2CGetFrequency(unsigned long BusType, unsigned long *pFrequency);

Inputs: unsigned long BusType
Bus being accessed. Currently, the only supported value is VL_I2C_BUS_TYPE_PRIMARY.

unsigned long *pFrequency
Pointer to the data buffer.

Outputs: Returns VL_API_OK on success.
VL_API_I2C_REQUESTED_BUS_NOT_FOUND if the requested bus is not found, or VL_API_I2C_READ_REGISTER_ERROR if an error occurred while attempting to retrieve the frequency of the bus.

VSL_I2CSetFrequency

Set the current frequency of the specified I^2C bus.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_I2CSetFrequency(unsigned long BusType, unsigned long Frequency);

Inputs: unsigned long BusType
Bus being accessed. Currently, the only supported value is VL_I2C_BUS_TYPE_PRIMARY.

unsigned long Frequency
Pointer to the data. Currently, the only supported values are
VL_I2C_FREQUENCY_100KHZ, or
VL_I2C_FREQENCY_400KHZ.

Outputs: Returns VL_API_OK on success.
VL_API_I2C_REQUESTED_BUS_NOT_FOUND if the requested bus is not found, or VL_API_I2C_WRITE_REGISTER_ERROR if an error occurred while attempting to retrieve the frequency of the bus.

SPI Calls

Access devices on the SPI bus. Many of these API calls require to include stdint ("#include <stdint.h>")

Table 11: SPI Parameters

	Parameter	Value
Clock Frequency	SPI_CLK_FREQ0	0.75Mhz (24Mhz/32)
	SPI_CLK_FREQ1	1.5 Mhz (24Mhz/16)
	SPI_CLK_FREQ2	2 Mhz (24 Mhz/8)
	SPI_CLK_FREQ2	6 Mhz (24Mhz/4)
Register Shift Direction	SPI_DIR_LEFT	Data is left-shifted (MSB first)
	SPI_DIR_RIGHT	Data is right-shifted (LSB first)

VSL_SPIIsAvailable

Loop continuously until the SPI bus becomes available or the loop counter expires.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_SPIIsAvailable()

Inputs: None

Outputs: Returns VL_API_OK on success.

VSL_SPISetFrequency

Select one of four clock frequencies for the SPI bus.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_SPISetFrequency(unsigned int Frequency)

Inputs: unsigned int Frequency

Frequency to be set for the SPI bus.

SPI_CLK_FREQ0 SPI_CLK_FREQ1
SPI_CLK_FREQ2 SPI_CLK_FREQ3

Outputs: Returns VL_API_OK on success.

VSL_SPISetShiftDirection

Controls the SPI shift direction from the SPIDATA(x) registers.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_SPISetShiftDirection(unsigned int Direction)

Inputs: unsigned int Direction

Register shift direction. Valid values are:
SPI_DIR_LEFT
SPI_DIR_RIGHT

Outputs: Returns VL_API_OK on success.

VSL_SPISetMode

Set the SPI bus mode.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_SPISetMode(unsigned int Mode)

Inputs: unsigned int Mode

SPI mode. Valid values are: 0, 1, 2, 3

Outputs: Returns VL_API_OK on success.

VSL_SPISetFrameSize

Set the SPI data frame size in number of bytes.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_SPISetFrameSize(unsigned int Size)

Inputs: unsigned int Size

SPI data frame size in number of bytes. Valid values are:
1 – 1 byte (8 bits)
2 – 2 bytes (16 bits)
3 – 3 bytes (24 bits)
4 – 4 bytes (32 bits)

Outputs: Returns VL_API_OK on success.
Returns VL_API_INVALID_ARG if Size is outside the allowed range.

VSL_SPIReadDataFrame

Read one frame of data from the data registers which is returned from the previous write operation to the same slave device.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_SPIReadDataFrame (uint32_t *pData);

Inputs: uint32_t *pData

Pointer to the destination data buffer. uint32_t defines a 32-bit or 4-byte buffer; add "#include <stdint.h>" to use it.

Outputs: Returns VL_API_OK on success.
Returns VL_API_SPI_READ_ERROR if an error occurs while attempting to read the register.

VSL_SPIWriteDataFrame

Write one frame of data to the data registers and initiate a SPI bus transaction to the specified slave device.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_SPIWriteDataFrame (unsigned int SlaveSelectNumber, unsigned char *pData);

Inputs: unsigned int SlaveSelectNumber

The SPI slave device selection number of the SPI device to communicate with. Currently the support values are VL_SPI_SS0 and VL_SPI_SS1.

uint32_t *pData

Pointer to the source data buffer. uint32_t defines a 32-bit or 4-byte buffer; add "#include <stdint.h>" to use it.

Outputs: Returns VL_API_OK on success.
Returns VL_API_SPI_WRITE_ERROR if an error occurs while attempting to write into the registers.

FPGA Calls

Use to access the onboard FPGA. The FPGA is mapped into I/O space on the LPC bus. Starting addresses are found in Table 38: Driver I/O Resource Assignments in the column titled “Linux Driver Setting”. Accessed all FPGA registers using the offset found in the Programmers Reference Manual for the particular board.

Note: Improper use of these functions could possibly lead to conditions preventing your system from booting. Use these commands with caution.

Examples: Register numbers should be specified as hexadecimal number, without leading “0x” characters.

Example 1. To address Register at offset 5 (TCR), use VSL_FPGAReadRegister(5);
Example 2. To address Register at offset 10 (MISCR1), use VSL_FPGAReadRegiser(10);

VSL_FPGAReadRegister

Read one byte of data from the specified register.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_FPGAReadRegister(unsigned long RegisterNumber, unsigned char *pData);

Inputs: unsigned long RegisterNumber
Number of the register to read. This value should be in hexadecimal, without leading 0x characters.

unsigned char *pData
Pointer to the destination buffer

Outputs: Returns VL_API_OK on success.
VL_API_ERROR if an error occurred while attempting to read the register.

VSL_FPGAWriteRegister

Write one byte of data to the specified register.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_FPGAWriteRegister(unsigned long RegisterNumber, unsigned char Data);

Inputs: unsigned long RegisterNumber
Number of the register to write into.

unsigned char Data
Data source buffer.

Outputs: Returns VL_API_OK on success.
VL_API_ERROR if an error occurred while attempting to read the register.

LCD Panel Control Calls

Use to query and control an LCD Panel. The API call VSL_LCDSetPanelName() is necessary for boards other than EPU-331x and EPU-4x6x.

VSL_LCDGetBacklight

Retrieve the LDC backlight brightness value.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_LCDGetBacklight(unsigned long * pBacklightValue);

Inputs: unsigned long *pBacklightValue
Pointer to the variable where the current backlight value will be stored.

Outputs: Returns VL_API_OK on success or
VL_API_VGA_BL_INTERFACE if the backlight value cannot be retrieved, or
VL_API_VGA_BL_VALUE_OUT_OF_RANGE if the retrieved value is not between the minimum backlight value (0) or the maximum backlight value.

VSL_LCDGetBacklightMaximum

Retrieve the LDC backlight brightness maximum value.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_LCDGetBacklightMaximum(unsigned long * pBacklightValue);

Inputs: unsigned long *pBacklightValue
Pointer to the variable where the maximum backlight value will be stored.

Outputs: Returns VL_API_OK on success or
VL_API_VGA_BL_INTERFACE if the backlight value cannot be retrieved.

VSL_LCDSetDisplayName

Set the LDC backlight display name. Name can be found in the /sys/class/backlight directory/folder.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_LCDSetDisplayName(char * pPanelName);

Inputs: char *pPanelName
Pointer to the variable where the panel name value is stored.

Outputs: Returns VL_API_OK on success or
VL_API_VGA_BL_DISPLAY_NOT_FOUND if the panel name cannot be found.

VSL_LCDSetBacklight

Set the LDC backlight brightness value.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_LCDSetBacklight(unsigned long backlightValue);

Inputs: unsigned long backlightValue
Value to set the backlight to. Value must be between VGA_BACKLIGHT_MIN (0) and VGA_BACKLIGHT_MAX (100).

Outputs: Returns VL_API_OK on success or
VL_API_VGA_BL_INTERFACE if the backlight value cannot be set, or
VL_API_VGA_BL_VALUE_OUT_OF_RANGE if the specified value is not between the minimum backlight value (0) or the maximum backlight value (100).

4 MPEu-C1E CAN

General API Information

Use the following API calls to configure, transmit and receive CAN packets using the CAN interface on the VersaLogic MPEu-C1E.

Find details in the “NXP Semiconductors UM10912 LPC546xx User Manual”, “Chapter 35 LPC546xx Controller Area Network Flexible Data” for the MCU used on this device. An overview of the necessary data to use the API is included in the following chapter.

The VersaLogic MPEu-C1E relies on API calls in the libusb library for communicating with the host device. libusb is a C library that provides generic access to USB devices. libusb uses a cross-platform API, requiring no special privileges or elevation to communicate with the device.

Many API calls return a VL_StatusT type return code. Successful calls return VL_API_OK. Unsuccessful calls return an appropriate value and requested data is not valid.

Operating Systems

The VersaLogic MPEu-C1E API will run under the following operating systems. For operating systems not listed below, please refer to the VersaLogic Software Support page:

- Ubuntu Linux (32 bit)
- Ubuntu Linux (64 bit)
- CentoOS Linux (64 bit)

Linux Installation

The following procedures are guidelines for installing VersaLogic MPEu-C1E API on a machine running a Linux operating system. Some modifications to the procedures may be required depending on the version of Linux running. Contact VersaLogic Technical Support if you encounter problems with the installation.

SETUP

1. Install and configure your VersaLogic MPEu-C1E mPCIe card onto the desired SBC board. A SBC board may contain only one MPEu-C1E.
2. Download the Linux VersaAPI MPEu-C1E package for your OS and extract its contents into a temporary directory. Make a note of the location of the temporary directory for access during these instructions.
3. Use the Linux installation script *vl_install_can.sh* to install the VersaAPI CAN shared library into the /usr/local/lib directory

PACKAGE CONTENTS

Table 12: Package content for MPEu-C1E Linux release

File Name	Function
vl_install_can.sh	VersaAPI MPEu-C1E install script
libVL_CAN.X.Y.Z.so	Shared library file for linking with applications
VL_OSALib.h	Header file containing all necessary VersaAPI definitions
VersaAPIGuid_vX.Y.Z.pdf	VersaAPI Installation and Reference Guide
Examples	Directory containing example code to access various features on a MPEu-C1E board. The example directory ‘can_sample’ contains an example of transmitting CAN messages using native Linux commands. The example directory ‘canVersaAPI_sample’ contains several examples for transmitting and receiving CAN messages using the VersaLogic API.

Updating the MCU

The following procedures are guidelines for updating the MCU on the VersaLogic MPEu-C1E using the Linux operating system. Two different methods are supplied. Some minor modifications to the procedures may be required depending on the version of Linux running. Contact VersaLogic Technical Support if you encounter problems with the installation.

SETUP

1. Install and configure your VersaLogic MPEu-C1E mPCIe card onto the desired SBC board. A SBC board may contain only one MPEu-C1E.
2. Download the Linux MPEu-C1E MCU update and extract its contents into a temporary directory. Make a note of the location of the temporary directory for access during these instructions.
3. Use the Linux installation script *mpeu-c1e_mcuUpdate.sh* to update the MCU on the MPEu-C1E.

PACKAGE CONTENTS

Table 13: Package content for MPEu-C1E Linux release

File Name	Function
mpeu-c1e_mcuUpdate.sh	VersaLogic MPEu-C1E MCU update script
mcuUpdate	Compiled application to retrieve MCU version information and put the MCU in update mode
Readme_mpeu-c1e.txt	Readme text file with current update instructions. If the update instructions in this document differ from update instructions in the Readme file, use the instructions in the Readme file.
MPEu-C1E-FW_*xxyyzz*.bin	MCU update file. xxyyzz refer to the version number of the MCU file.

UPGRADE INSTRUCTIONS OVER USB

To update the MPEu-C1E MCU using Linux running on the carrier board, you need the following:

- Supported Linux OS installed on the carrier board (i.e. Ubuntu 16.04 LTS, or later)
- dfu-util installed (apt-get install dfu-util)
- The VersaLogic utility mcuUpdate to put the MCU into update mode
- The VersaLogic MPEu-C1e API library libVL_CAN.1.0.0.so, or later

Tool Installation Instructions

libVL_CAN.X.Y.Z.so

- Must be copied to /usr/local/lib

dfu-util application

- apt-get install dfu-util
- Tools should be in the Linux execution path ($PATH)

Update Instructions

From a carrier shell window execute the following command

mpeu-c1e_mcuUpdate.sh -f MPEu-C1E-FW_*xxyyzz*.bin

Where *xxyyzz*.bin is the MCU update file included in the release package.

UPGRADE INSTRUCTIONS OVER SWD

Instructions on how to update the MCU FW on the MPEu-C1E with a J-Link probe:

- Obtain a J-Link probe and the J-Link programming software from https://www.segger.com/downloads/jlink/
- Connect the J-Link probe's USB port to a PC with the above SW installed
- Connect the other end to the J2 header on the MPEu-C1E
- Run the "J-Link Commander" application on the PC and enter
    - connect
    - Use the device LPC54608J512 (should be default)
    - Specify SWD as the target interface
    - Use default speed of 4000 KHz
    - Once the J-Link Commander says it connected, enter "loadfile <filename>" to load the attached file onto the MCU flash.

CAN Data Types

TX buffer type is used to configure the Tx buffer for sending CAN packets.

Table 14: T0 bit description - MCAN_TX_PACKET

Bit	Symbol	Value	Description
28:0	ID	-	Identifier. Standard or extended identifier depending on the XTD bit. A standard identifier is stored into bits 28:18.
29	RTR		Remote transmission request.
		0	Transmit data frame
		1	Transmit remote frame
30	XTD		Extended identifier
		0	11-bit standard identifier
		1	29-bit extended identifier
31	ESI		Error state identifier
		0	ESI bit in CAN FD format depends only on error passive flag
		1	ESI bit in CAN FD format transmitted recessive

Table 15: T1 bit description – MCAN_TX_PACKET

Bit	Symbol	Value	Description
15:0	-	-	Reserved

19:16	DLC		Data length code 0 – 8 = CAN + CAN FD: transmit frame has 0 - 8 data bytes 9 – 15 = CAN: transmit frame has 8 data bytes 9 – 15 = CAN FD: transmit frame has 12/16/20/24/32/48/64 data bytes
20	BRS		Bit rate switching
		0	CAN FD frames transmitted without bit rate switching
		1	CAN FD frames transmitted with bit rate switching
21	FDF		FD format
		0	Frame transmitted in classic CAN format
		1	Frame transmitted in CAN FD format
22	-	-	Reserved
23	EFC		Event FIFO control
		0	Do not store Tx events
		1	Store Tx events
31:24	MM	-	Written during Tx buffer configuration. Copied into Tx event FIFO element for identification of Tx message status

Table 16: PCIe board identifier - VL_CANBoardT

Parameter Name	Value
VL_CAN_BOARD0	Default. Only supported value at this time.
VL_CAN_BOARD1	Not supported at this time.

Table 17: CAN port identifier - VL_CANPortT

Parameter Name	Value
VL_CAN_PORT0	CAN port 0.

VL_CAN_PORT1	CAN port 1.

```
typedef struct _mcan_tx_packet_ {
   uint31_t          t0;
   uint31_t          t1;
   uint7_t           mcanTxData[ CAN_MAX_FRAME_SIZE ];
} MCAN_TX_PACKET;
```

CAN API Calls

CAN PORT CALLS

Use these API calls to open, close, identify and get status of a CAN port.

VSL_CanOpenPort

Opens the specified CAN port for transmitting and/or receiving CAN packets. Must be called and have a return value of VL_API_OK for all other CAN API calls to work as expected. See Appendix D for supported arbitration and data bit rates. For arbitration/data bit rates not supported with this API call, see VSL_CanOpenPortWithUserTiming() for an alternative.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_CANOpenPort(VL_CANBoardT boardID, VL_CANPortT portID, uint32_t arbitrationBitRate, uint32_t dataBitRate, uint32_t rxEnabled, uint32_t loopbackEnabled);

Inputs: VL_CANBoardT boardID
PCIe CAN board identifier. Valid values are VL_CAN_BOARD0 and VL_CAN_BOARD1. Currently only one MPEu-C1E, VL_CAN_BOARD0, is allowed on a carrier board at this time.

VL_CANPortT portID
Name of the CAN port to open. Valid values are VL_CAN_PORT0 and VL_CAN_PORT1.

uint32_t arbitrationBitRate

Nominal bit rate in bits/second. For classical CAN and CAN-FD frames that do not use bit rate switching this is the speed the frames will be transmitted at. Typical values are 1000000 (1 Mbits/s), 500000 (500 Kbits/s) and 250000 (250 Kbits/s).

uint32_t dataBitRate

Bit rate for the data byte and checksum in bit/second. Only used for CAN-FD frames that use bit rate swtiching. Set to zero to disable bit rate switching. Must be higher than nominalBaudRate. Typical values are 2000000 (2 Mbits/s), 4000000 (4 Mbits/s), 8000000 (8 Mbits/s) and 10 (10000000 Mbits/s).

uint8_t rxEnabled

Enable receiving packet after it has been transmitted. Valid values are VL_ENABLE_RX_ON_TX (0x1) and VL_DISABLE_RX_ON_TX (0x0).

uint8_t loopbackEnabled

Enable loopback mode. Valid values are VL_ENABLE_LOOPBACK (0x1) and VL_DISABLE_LOOPBACK (0x0).

Outputs: Returns VL_API_OK on success, or
VL_API_INVALID_ARG if a function argument is invalid,
VL_API_CAN_OPEN_DEVICE_ERROR if the specified CAN board cannot be opened, VL_API_CAN_OPEN_PORT_ERROR if the specified CAN port cannot be opened, or VL_API_CAN_NO_DEVICE_FOUND if the CAN device cannot be found on the PCIe bus.

VSL_CanOpenPortWithUserTiming

Opens the specified CAN port for transmitting and/or receiving CAN packets. Must be called and have a return value of VL_API_OK for all other CAN API calls to work as expected. See Appendix D for supported arbitration and data bit rates. This API call can be used for arbitration/data bit rates not listed in Appendix D, or if a user wants to specify their own timing.

Syntax:
```
VL_OSALIB_API VL_APIStatusT
VSL_CANOpenPortWithUserTiming(VL_CANBoardT  boardID,
VL_CANPortT portID,
uint32_t arbitrationBitRate, uint32_t dataBitRate,
uint8_t rxEnabled, uint8_t loopbackEnabled,
uint32_t masterDivider,
uint16_t arbPreDivider, uint8_t arbResyncJumpWidth,
uint8_t arbSeg1, uint8_t arbSeg2,
uint16_t dataPreDivider, uint8_t dataResyncJumpWidth,
uint8_t dataSeg1, uint8_t dataSeg2);
```

Inputs: VL_CANBoardT boardID
PCIe CAN board identifier. Valid values are VL_CAN_BOARD0 and VL_CAN_BOARD1. Currently only one MPEu-C1E, VL_CAN_BOARD0, is allowed on a carrier board at this time.

VL_CANPortT portID
Name of the CAN port to open. Valid values are VL_CAN_PORT0 and VL_CAN_PORT1.

uint32_t arbitrationBitRate

Nominal bit rate in bits/second. For classical CAN and CAN-FD frames that do not use bit rate switching this is the speed the frames will be transmitted at. Typical values are 1000000 (1 Mbits/s), 500000 (500 Kbits/s) and 250000 (250 Kbits/s).

uint32_t dataBitRate

Bit rate for the data byte and checksum in bit/second. Only used for CAN-FD frames that use bit rate swtiching. Set to zero to disable bit rate switching. Must be higher than nominalBaudRate. Typical values are 2000000 (2 Mbits/s), 4000000 (4 Mbits/s), 8000000 (8 Mbits/s) and 10 (10000000 Mbits/s).

uint8_t rxEnabled

Enable receiving packet after it has been transmitted. Valid values are VL_ENABLE_RX_ON_TX (0x1) and VL_DISABLE_RX_ON_TX (0x0).

uint8_t loopbackEnabled

Enable loopback mode. Valid values are VL_ENABLE_LOOPBACK (0x1) and VL_DISABLE_LOOPBACK (0x0).

uint32_t masterDivider

Master clock divider.

uint16_t arbPreDivider

Arbitration clock pre-scalar division factor.

uint8_t arbResyncJumpWidth

Arbitration Re-synce Jump Width.

uint8_t arbSeg1

Arbitration Data Time Segment 1.

uint8_t arbSeg2

Arbitration Data Time Segment 2.

uint16_t dataPreDivider

Arbitration clock pre-scalar division factor.

uint8_t dataResyncJumpWidth

Re-synce Jump Width.

uint8_t dataSeg1

Data Time Segment 1.

uint8_t dataSeg2

Data Time Segment 2.

Outputs: Returns VL_API_OK on success, or VL_API_INVALID_ARG if a function argument is invalid, VL_API_CAN_OPEN_DEVICE_ERROR if the specified CAN board cannot be opened, VL_API_CAN_OPEN_PORT_ERROR if the specified CAN port cannot be opened, or VL_API_CAN_NO_DEVICE_FOUND if the CAN device cannot be found on the PCIe bus.

VSL_CANClosePort

Closes the specified CAN port if opened.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_CANClosePort(VL_CANBoardT boardID, VL_CANPortT portID);

Inputs: VL_CANBoardT boardID
PCIe CAN board identifier. Valid values are VL_CAN_BOARD0 and VL_CAN_BOARD1. Currently only one MPEu-C1E, VL_CAN_BOARD0, is allowed on a carrier board.

VL_CANPortT portI
Name of the CAN port to open. Valid values are VL_CAN_PORT0 and VL_CAN_PORT1.

Outputs: Returns VL_API_OK on success, or VL_API_INVALID_ARG if a function argument is invalid.

VSL_CANGetPortStatus

Retrieve the port status for the named CAN port.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_CANGetPortStatus(VL_CANBoardT boardID, VL_CANPortT portID, VL_CANPortStatusT *pCanPortStatus);

Inputs: VL_CANBoardT boardID
PCIe CAN board identifier. Valid values are VL_CAN_BOARD0 and VL_CAN_BOARD1. Currently only one MPEu-C1E, VL_CAN_BOARD0, is allowed on a carrier board.

VL_CANPortT portID

Name of the CAN port to open. Valid values are VL_CAN_PORT0 and VL_CAN_PORT1.

Outputs:
VL_CANPortStatusT *pCanPortStatus
VL_API_CAN_PORT_OPEN or VL_API_CAN_PORT_CLOSED.

Returns VL_API_OK on success, or
VL_API_INVALID_ARG if a function argument is invalid,
VL_API_CAN_OPEN_DEVICE_ERROR if the specified CAN board is not enabled, VL_API_CAN_OPEN_PORT_ERROR if the specified CAN port is not opened.

MCU VERSION CALLS

Use these API calls to retrieve and update the MCU firmware on the MPEu-C1E.

VSL_CANUpdateMCUFwVersion

Update the MPEu-C1E MCU firmware version.

Syntax: VL_OSALIB_API VL_APIStatusT
VSL_CANUpdateMCUFwVersion(VL_CANBoardT boardID);

Inputs: VL_CANBoardT boardID
PCIe CAN board identifier. Valid values are VL_CAN_BOARD0 and VL_CAN_BOARD1. Currently only one MPEu-C1E, VL_CAN_BOARD0, is allowed on a carrier board.

Outputs: Returns VL_API_OK on success, or
VL_API_INVALID_ARG if a function argument is invalid, VL_API_ERROR for other errors.

VSL_CANGetMCUFwVersion

Retrieve the MPEu-C1E MCU firmware version.

Syntax: VL_OSALIB_API VL_APIStatusT
VSL_CANGetMCUFwVersion(VL_CANBoardT boardID, uint32_t *pMcuFwVer);

Inputs: VL_CANBoardT boardID
PCIe CAN board identifier. Valid values are VL_CAN_BOARD0 and VL_CAN_BOARD1. Currently only one MPEu-C1E, VL_CAN_BOARD0, is allowed on a carrier board.

Outputs: uint32_t *pMcuFwVer will contain the value of the MCU firmware revision

Bits 5:4 Major release number; Bits 3:2 Minor release number; Bits 1:0 Release release number.

Returns VL_API_OK on success, or
VL_API_INVALID_ARG if a function argument is invalid, VL_API_ERROR for other errors.

TRANSMITTING CAN PACKETS

Use this API call to transmit a CAN packet out a specified port.

VSL_CANTransmit

Transmit the specified packet out the specified CAN port.

Syntax: VL_OSALIB_API VL_APIStatusT VSL_CANTransmit(VL_CANBoardT boardID, VL_CANPortT portID, MCAN_TX_PACKET *pTxPacket);

Inputs: VL_CANBoardT boardID
PCIe CAN board identifier. Valid values are VL_CAN_BOARD0 and VL_CAN_BOARD1. Currently only one MPEu-C1E, VL_CAN_BOARD0, is allowed on a carrier board.

VL_CANPortT portID
Name of the CAN port to open. Valid values are VL_CAN_PORT0 and VL_CAN_PORT1.

MCAN_TX_PACKET *pTxPacket
Pointer to a MCAN_TX_PACKET structure.

Outputs: Returns VL_API_OK on success, or
VL_API_INVALID_ARG if a function argument is invalid, VL_API_CAN_OPEN_DEVICE_ERROR if the specified CAN board is not enabled, VL_API_CAN_OPEN_PORT_ERROR if the specified CAN port is not opened, VL_API_CAN_NO_DEVICE_FOUND if the CAN device cannot be found on the PCIe bus, or VL_API_CAN_TX_ERROR if there is an error transmitting the packet.

RECEIVING CAN PACKETS

When the MPEu-C1E receives a CAN packet on the USB interface the 'user_CANRxCallBackFunction()' is called. This function can be defined in your application code with the follow header:

```
void user_CANRxCallBackFunction(VL_USB_CAN_XFER *pRxPacket, int txStatus);
```

where VL_USB_CAN_XFER is defined in VL_OSALib.h as:

```
typedef struct _usb_can_packet_ {
   uint8_t          size;
   uint8_t          cmd;
   uint8_t          port;
   uint8_t          misc;
   uint32_t         extendedStatus;
   union {
      MCAN_OPEN              openInfo;
      MCAN_RX_PACKET     rxPacket;
      MCAN_TX_PACKET     txPacket;
      MCAN_TX_DONE         txEvent;
   } u;
} VL_USB_CAN_XFER;
```

The 'pRxPacket->cmd' will be set to USB_CAN_CMD_RX when a CAN packet is receive. A 'txStatus' of LIBUSB_TRANSFER_COMPLETED is set when the usb transfer has completed. See documentation for libusb for a complete list of values for txStatus. Once the 'pRxPacket->cmd' and 'txStatus' are checked appropriately, the callback function can process the packet as a valid CAN packet.

MCU REGISTER ACCESSING

Use these API calls to read certain MCU registers.

VSL_CANGetProtocolRegisterStatus

Retrieve the MPEu-C1E MCU's Protocol Status Register (PSR) value. Returns VL_API_OK on success, or VL_API_INVALID_ARG if a function argument is invalid, VL_API_ERROR for other errors.

Syntax: VL_OSALIB_API VL_APIStatusT
VSL_CANGetProtocolRegisterStatus(VL_CANBoardT boardID,
uint32_t *pLEC, uint32_t *pACT, uint32_t *pEP, uint32_t *pEW,
uint32_t *pBO, uint32_t *pDLEC, uint32_t *pRESI, uint32_t *pRBRS,
uint32_t *pRFDF, uint32_t *pPXE, uint32_t *pTDCV);

Inputs: VL_CANBoardT boardID
PCIe CAN board identifier. Valid values are VL_CAN_BOARD0 and VL_CAN_BOARD1. Currently only one MPEu-C1E, VL_CAN_BOARD0, is allowed on a carrier board.

Outputs: uint32_t *pLEC will contain the value of the MCU PSR, bits [2:0].
uint32_t *pACT will contain the value of the MCU PRS, bits [4:3].
uint32_t *EP will contain the value of the MCU PSR, bit [5].
uint32_t *pEW will contain the value of the MCU PSR, bit [6].
uint32_t *pBO will contain the value of the MCU PSR, bit [7].
uint32_t *pDLEC will contain the value of the MCU PSR, bits [10:8].
uint32_t *pRESI will contain the value of the MCU PSR, bit [11].
uint32_t *pRBRS will contain the value of the MCU PSR, bit [12].
uint32_t *pRBRS will contain the value of the MCU PSR, bit [12].
uint32_t *pRFDR will contain the value of the MCU PSR, bit [13].
uint32_t *pPXE will contain the value of the MCU PSR, bit [14].
uint32_t *pTDCV will contain the value of the MCU PSR, bit [22:16].

VSL_CANGetErrorCounterRegisterStatus

Retrieve the MPEu-C1E MCU's Error Counter Register (ECR) value. Returns VL_API_OK on success, or VL_API_INVALID_ARG if a function argument is invalid, VL_API_ERROR for other errors.

Syntax: VL_OSALIB_API VL_APIStatusT
VSL_CANGetErrorCounterRegisterStatus(VL_CANBoardT boardID,
uint32_t *pTEC, uint32_t *pREC, uint32_t *pRP, uint32_t *pCEL);

Inputs: VL_CANBoardT boardID
PCIe CAN board identifier. Valid values are VL_CAN_BOARD0 and VL_CAN_BOARD1. Currently only one MPEu-C1E, VL_CAN_BOARD0, is allowed on a carrier board.

Outputs: uint32_t *pTEC will contain the value of the MCU ECR, bits [7:0].
uint32_t *pREC will contain the value of the MCU ECR, bits [14:8].
uint32_t *RP will contain the value of the MCU ECR, bit [15].
uint32_t *pCEL will contain the value of the MCU ECR, bit [23:16].

Appendix A. Channel Assignment Tables

The tables in this appendix list the digital I/O (DIO/GPIO), 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

Table 18: EBX/EPIC 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	EBX-18 J5 pin	EBX-38 J21Pin
0	00	DIO_CHANNEL_1	1	1	1	16	1	1
1	01	DIO_CHANNEL_2	2	2	2	17	2	2
2	02	DIO_CHANNEL_3	3	3	3	18	3	3
3	03	DIO_CHANNEL_4	4	4	4	19	4	4
4	04	DIO_CHANNEL_5	6	6	6	21	6	6
5	05	DIO_CHANNEL_6	7	7	7	22	7	7
6	06	DIO_CHANNEL_7	8	8	8	23	8	8
7	07	DIO_CHANNEL_8	9	9	9	24	9	9
8	08	DIO_CHANNEL_9	11	11	11	26	11	11
9	09	DIO_CHANNEL_10	12	12	12	27	12	12
10	0A	DIO_CHANNEL_11	13	13	13	28	13	13
11	0B	DIO_CHANNEL_12	14	14	14	29	14	14
12	0C	DIO_CHANNEL_13	16	16	16	31	16	16
13	0D	DIO_CHANNEL_14	17	17	17	32	17	17
14	0E	DIO_CHANNEL_15	18	18	18	33	18	18
15	0F	DIO_CHANNEL_16	19	19	19	34	19	19
								J26 Pin
16	10	DIO_CHANNEL_17	21	21	21	—	21	1
17	11	DIO_CHANNEL_18	22	22	22	—	22	2
18	12	DIO_CHANNEL_19	23	23	23	—	23	3
19	13	DIO_CHANNEL_20	24	24	24	—	24	4
20	14	DIO_CHANNEL_21	26	26	26	—	26	6
21	15	DIO_CHANNEL_22	27	27	27	—	27	7
22	16	DIO_CHANNEL_23	28	28	28	—	28	8
23	17	DIO_CHANNEL_24	29	29	29	—	29	9
24	18	DIO_CHANNEL_25	31	31	31	—	31	11
25	19	DIO_CHANNEL_26	32	32	32	—	32	12
26	1A	DIO_CHANNEL_27	33	33	33	—	33	13
27	1B	DIO_CHANNEL_28	34	34	34	—	34	14
28	1C	DIO_CHANNEL_29	36	36	36	—	36	16
29	1D	DIO_CHANNEL_30	37	37	37	—	37	17
30	1E	DIO_CHANNEL_31	38	38	38	—	38	18
31	1F	DIO_CHANNEL_32	39	39	39	—	39	19
								J4 Pin
32	20	DIO_CHANNEL_33	—	—	—	—	—	27
33	21	DIO_CHANNEL_34	—	—	—	—	—	28
34	22	DIO_CHANNEL_35	—	—	—	—	—	29
35	23	DIO_CHANNEL_36	—	—	—	—	—	30
36	24	DIO_CHANNEL_37	—	—	—	—	—	33
37	25	DIO_CHANNEL_38	—	—	—	—	—	34
38	26	DIO_CHANNEL_39	—	—	—	—	—	35
39	27	DIO_CHANNEL_40	—	—	—	—	—	36

Table 19: PC/104-Plus/PCIe/104 Board On-board DIO Channels

Channel Number	Hex Code	Channel Name	EPMe-30 J21 Pin	EPMe-42 J3 Pin	EPM-31 J21 Pin	EPM-19 J9 Pin	EPM-39
0	00	DIO_CHANNEL_1	1	27	1	27	—
1	01	DIO_CHANNEL_2	2	28	2	28	—
2	02	DIO_CHANNEL_3	3	29	3	29	—
3	03	DIO_CHANNEL_4	4	30	4	30	—
4	04	DIO_CHANNEL_5	6	33	6	33	—
5	05	DIO_CHANNEL_6	7	34	7	34	—
6	06	DIO_CHANNEL_7	8	35	8	35	—
7	07	DIO_CHANNEL_8	9	36	9	36	—
8	08	DIO_CHANNEL_9	11	—	11	—	—
9	09	DIO_CHANNEL_10	12	—	12	—	—
10	0A	DIO_CHANNEL_11	13	—	13	—	—
11	0B	DIO_CHANNEL_12	14	—	14	—	—
12	0C	DIO_CHANNEL_13	16	—	16	—	—
13	0D	DIO_CHANNEL_14	17	—	17	—	—
14	0E	DIO_CHANNEL_15	18	—	18	—	—
15	0F	DIO_CHANNEL_16	19	—	19	—	—
					J4 Pin		**J4 Pin**
16	10	DIO_CHANNEL_17	—	—	27	—	27
17	11	DIO_CHANNEL_18	—	—	28	—	28
18	12	DIO_CHANNEL_19	—	—	29	—	29
19	13	DIO_CHANNEL_20	—	—	30	—	30
20	14	DIO_CHANNEL_21	—	—	33	—	33
21	15	DIO_CHANNEL_22	—	—	34	—	34
22	16	DIO_CHANNEL_23	—	—	35	—	35
23	17	DIO_CHANNEL_24	—	—	36	—	36

Table 20: EPU Board On-board DIO Channels

Channel Number	Hex Code	Channel Name	EPU-3311 J21 pin	EPU-3312 J21 Pin	EPU-4011 J21 pin	EPU-4012 J21 Pin	EPU-4X62 J30 Pin
0	00	DIO_CHANNEL_1	27	27	27	27	1
1	01	DIO_CHANNEL_2	28	28	28	28	2
2	02	DIO_CHANNEL_3	29	29	29	29	3
3	03	DIO_CHANNEL_4	30	30	30	30	4
4	04	DIO_CHANNEL_5	33	33	33	33	6
5	05	DIO_CHANNEL_6	34	34	34	34	7
6	06	DIO_CHANNEL_7	35	35	35	35	8
7	07	DIO_CHANNEL_8	36	36	36	36	9
8	08	DIO_CHANNEL_9	—	—	—	—	11
9	09	DIO_CHANNEL_10	—	—	—	—	12
10	0A	DIO_CHANNEL_11	—	—	—	—	13
11	0B	DIO_CHANNEL_12	—	—	—	—	14
12	0C	DIO_CHANNEL_13	—	—	—	—	16
13	0D	DIO_CHANNEL_14	—	—	—	—	17
14	0E	DIO_CHANNEL_15	—	—	—	—	18
15	0F	DIO_CHANNEL_16	—	—	—	—	19

							J2 Pin
16	10	DIO_CHANNEL_17	—	—	—	—	27
17	11	DIO_CHANNEL_18	—	—	—	—	28
18	12	DIO_CHANNEL_19	—	—	—	—	29
19	13	DIO_CHANNEL_20	—	—	—	—	30
20	14	DIO_CHANNEL_21	—	—	—	—	33
21	15	DIO_CHANNEL_22	—	—	—	—	34
22	16	DIO_CHANNEL_23	—	—	—	—	35
23	17	DIO_CHANNEL_24	—	—	—	—	36

Table 21: 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_1	1
65	41	DIO_SS0_CHANNEL_2	2
66	42	DIO_SS0_CHANNEL_3	3
67	43	DIO_SS0_CHANNEL_4	4
			J3 Pin
68	44	DIO_SS0_CHANNEL_5	1
69	45	DIO_SS0_CHANNEL_6	2
70	46	DIO_SS0_CHANNEL_7	3
71	47	DIO_SS0_CHANNEL_8	4
			J4 Pin
72	48	DIO_SS0_CHANNEL_9	1
73	49	DIO_SS0_CHANNEL_10	2
74	4A	DIO_SS0_CHANNEL_11	3
75	4B	DIO_SS0_CHANNEL_12	4
			J5 Pin
76	4C	DIO_SS0_CHANNEL_13	1
77	4D	DIO_SS0_CHANNEL_14	2
78	4E	DIO_SS0_CHANNEL_15	3
79	4F	DIO_SS0_CHANNEL_16	4

Table 22: 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_SS1_CHANNEL_1	1
81	51	DIO_SS1_CHANNEL_2	2
82	52	DIO_SS1_CHANNEL_3	3
83	53	DIO_SS1_CHANNEL_4	4
			J3 Pin
84	54	DIO_SS1_CHANNEL_5	1
85	55	DIO_SS1_CHANNEL_6	2
86	56	DIO_SS1_CHANNEL_7	3
87	57	DIO_SS1_CHANNEL_8	4
			J4 Pin
88	58	DIO_SS1_CHANNEL_9	1
89	59	DIO_SS1_CHANNEL_10	2
90	5A	DIO_SS1_CHANNEL_11	3
91	5B	DIO_SS1_CHANNEL_12	4
			J5 Pin
92	5C	DIO_SS1_CHANNEL_13	1
93	5D	DIO_SS1_CHANNEL_14	2
94	5E	DIO_SS1_CHANNEL_15	3
95	5F	DIO_SS1_CHANNEL_16	4

Table 23: 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_SS2_CHANNEL_1	1
97	61	DIO_SS2_CHANNEL_2	2
98	62	DIO_SS2_CHANNEL_3	3
99	63	DIO_SS2_CHANNEL_4	4
			J3 Pin
100	64	DIO_SS2_CHANNEL_5	1
101	65	DIO_SS2_CHANNEL_6	2
102	66	DIO_SS2_CHANNEL_7	3
103	67	DIO_SS2_CHANNEL_8	4
			J4 Pin
104	68	DIO_SS2_CHANNEL_9	1
105	69	DIO_SS2_CHANNEL_10	2
106	6A	DIO_SS2_CHANNEL_11	3
107	6B	DIO_SS2_CHANNEL_12	4
			J5 Pin
108	6C	DIO_SS2_CHANNEL_13	1
109	6D	DIO_SS2_CHANNEL_14	2
110	6E	DIO_SS2_CHANNEL_15	3
111	6F	DIO_SS2_CHANNEL_16	4

Table 24: 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_SS3_CHANNEL_1	1
113	71	DIO_SS3_CHANNEL_2	2
114	72	DIO_SS3_CHANNEL_3	3
115	73	DIO_SS3_CHANNEL_4	4
			J3 Pin
116	74	DIO_SS3_CHANNEL_5	1
117	75	DIO_SS3_CHANNEL_6	2
118	76	DIO_SS3_CHANNEL_7	3
119	77	DIO_SS3_CHANNEL_8	4
			J4 Pin
120	78	DIO_SS3_CHANNEL_9	1
121	79	DIO_SS3_CHANNEL_10	2
122	7A	DIO_SS3_CHANNEL_11	3
123	7B	DIO_SS3_CHANNEL_12	4
			J5 Pin
124	7C	DIO_SS3_CHANNEL_13	1
125	7D	DIO_SS3_CHANNEL_14	2
126	7E	DIO_SS3_CHANNEL_15	3
127	7F	DIO_SS3_CHANNEL_16	4

Table 25: 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

Table 26: PCIe Expansion DIO Channels – VL-MPEe-U2 (Note 2)

Channel Number	Hex Code	Channel Name	VL-MPEe-U2 J3 Pin
160	A0	DIO_U2_CHANNEL_1	1
161	A1	DIO_U2_CHANNEL_2	2
162	A2	DIO_U2_CHANNEL_3	3
163	A3	DIO_U2_CHANNEL_4	4
164	A4	DIO_U2_CHANNEL_5	6
165	A5	DIO_U2_CHANNEL_6	7
166	A6	DIO_U2_CHANNEL_7	8
167	A7	DIO_U2_CHANNEL_8	9
168	A8	DIO_U2_CHANNEL_9	11
169	A9	DIO_U2_CHANNEL_10	12
170	AA	DIO_U2_CHANNEL_11	13
171	AB	DIO_U2_CHANNEL_12	14
172	AC	DIO_U2_CHANNEL_13	15 (Note 1)

Note 1: This pin is Ground on standard products but can be made an I/O line on custom products.
Note 2: When using the VL_MPEe-U2, drivers from Exar are required. To install drivers for the VL-MPEe-U2, go to the VL-MPEe-U2 Support Page.

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

Table 27: 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	EPMe-30 J19 Pin	EPM-31 J21 Pin
0	00	SPI_A0_CHANNEL_1	21	21	21	11	21	21
1	01	SPI_A0_CHANNEL_2	22	22	22	12	22	22
2	02	SPI_A0_CHANNEL_3	23	23	23	13	23	23
3	03	SPI_A0_CHANNEL_4	24	24	24	14	24	24
4	04	SPI_A0_CHANNEL_5	26	26	26	—	26	26
5	05	SPI_A0_CHANNEL_6	27	27	27	—	27	27
6	06	SPI_A0_CHANNEL_7	28	28	28	—	28	28
7	07	SPI_A0_CHANNEL_8	29	29	29	—	29	29

Table 28: VL-SPX-4 Expansion Board DAC Channels – Using Slave Select 0

Channel Number	Hex Code	Channel Name	VL-SPX-4 J2 Pin
64	40	SPX_A0_SS0_CHANNEL_1	1
65	41	SPX_A0_SS0_CHANNEL_2	2
66	42	SPX_A0_SS0_CHANNEL_3	3
67	43	SPX_A0_SS0_CHANNEL_4	4

Table 29: VL-SPX-4 Expansion Board DAC Channels – Using Slave Select 1

Channel Number	Hex Code	Channel Name	VL-SPX-4 J2 Pin
80	50	SPX_A0_SS1_CHANNEL_1	1
81	51	SPX_A0_SS1_CHANNEL_2	2
82	52	SPX_A0_SS1_CHANNEL_3	3
83	53	SPX_A0_SS1_CHANNEL_4	4

Table 30: VL-SPX-4 Expansion Board DAC Channels – Using Slave Select 2

Channel Number	Hex Code	Channel Name	VL-SPX-4 J2 Pin
96	60	SPX_A0_SS2_CHANNEL_1	1
97	61	SPX_A0_SS2_CHANNEL_2	2
98	62	SPX_A0_SS2_CHANNEL_3	3
99	63	SPX_A0_SS2_CHANNEL_4	4

Table 31: VL-SPX-4 Expansion Board DAC Channels – Using Slave Select 3

Channel Number	Hex Code	Channel Name	VL-SPX-4 J2 Pin
112	70	SPX_A0_SS3_CHANNEL_1	1
113	71	SPX_A0_SS3_CHANNEL_2	2
114	72	SPX_A0_SS3_CHANNEL_3	3
115	73	SPX_A0_SS3_CHANNEL_4	4

Analog to Digital (ADC) Channel Assignments

The definitions below are 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

Table 32: CPU Board On-board ADC Channels

Channel Number	Hex Code	Channel Name	Pin Number						
			EBX-37 J22	EBX-41 J19	EPIC-17 J18	EPIC-25 J25	EPMe-30 J19	EPM-31 J21	EPU-3312 J19
0	00	SPI_AI_CHANNEL_1	1	1	1	1	1	1	1
1	40	SPI_AI_CHANNEL_2	2	2	2	2	2	2	2
2	10	SPI_AI_CHANNEL_3	3	3	3	3	3	3	3
3	50	SPI_AI_CHANNEL_4	4	4	4	4	4	4	4
4	20	SPI_AI_CHANNEL_5	6	6	6	6	6	6	6
5	60	SPI_AI_CHANNEL_6	7	7	7	7	7	7	7
6	30	SPI_AI_CHANNEL_7	8	8	8	8	8	8	8
7	70	SPI_AI_CHANNEL_8	9	9	9	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	—	—	—	—

Table 33: VL-SPX-1 Expansion Board ADC Channels – Using Slave Select 0

Channel Number	Hex Code	Channel Name	VL-SPX-1 J2 Pin
64	04	SPX_AI_SS0_CHANNEL_1	1
65	14	SPX_AI_SS0_CHANNEL_2	2
66	24	SPX_AI_SS0_CHANNEL_3	3
67	34	SPX_AI_SS0_CHANNEL_4	4
			J3 Pin
68	44	SPX_AI_SS0_CHANNEL_5	1
69	54	SPX_AI_SS0_CHANNEL_6	2
70	64	SPX_AI_SS0_CHANNEL_7	3
71	74	SPX_AI_SS0_CHANNEL_8	4

Table 34: VL-SPX-1 Expansion Board ADC Channels – Using Slave Select 1

Channel Number	Hex Code	Channel Name	VL-SPX-1 J2 Pin
80	05	SPX_AI_SS1_CHANNEL_1	1
81	45	SPX_AI_SS1_CHANNEL_2	2
82	15	SPX_AI_SS1_CHANNEL_3	3
83	55	SPX_AI_SS1_CHANNEL_4	4
			J3 Pin
84	25	SPX_AI_SS1_CHANNEL_5	1
85	65	SPX_AI_SS1_CHANNEL_6	2
86	35	SPX_AI_SS1_CHANNEL_7	3
87	75	SPX_AI_SS1_CHANNEL_8	4

Table 35: VL-SPX-1 Expansion Board ADC Channels – Using Slave Select 2

Channel Number	Hex Code	Channel Name	VL-SPX-1 J2 Pin
96	06	SPX_AI_SS2_CHANNEL_1	1
97	46	SPX_AI_SS2_CHANNEL_2	2
98	16	SPX_AI_SS2_CHANNEL_3	3
99	56	SPX_AI_SS2_CHANNEL_4	4
			J3 Pin
100	26	SPX_AI_SS2_CHANNEL_5	1
101	66	SPX_AI_SS2_CHANNEL_6	2
102	36	SPX_AI_SS2_CHANNEL_7	3
103	76	SPX_AI_SS2_CHANNEL_8	4

Table 36: VL-SPX-1 Expansion Board ADC Channels – Using Slave Select 3

Channel Number	Hex Code	Channel Name	VL-SPX-1 J2 Pin
112	07	SPX_AI_SS3_CHANNEL_1	1
113	47	SPX_AI_SS3_CHANNEL_2	2
114	17	SPX_AI_SS3_CHANNEL_3	3
115	57	SPX_AI_SS3_CHANNEL_4	4
			J3 Pin
116	27	SPX_AI_SS3_CHANNEL_5	1
117	67	SPX_AI_SS3_CHANNEL_6	2
118	37	SPX_AI_SS3_CHANNEL_7	3
119	77	SPX_AI_SS3_CHANNEL_8	4

Table 37: 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

Appendix B. Changing Driver I/O Resources

Table 38: Driver I/O Resource Assignments

Board	**Windows Driver Setting** (Under the VersaLogic SPX Driver Properties "Resource" Tab)	**Linux Driver Setting** (vldrive FPGA_BASE command line option)
EPIC-25 EBX-37 EBX-41 EBX-18 EPM-19 MPEe-A1/A2 MPEe-U2	Basic Configuration 0001 (default)	0xCA0 (Default)
EPMe-30 EPM-31 EPM-39 EPMe-42 EPMe-51 EPMe-5120 EPM-43 EBX-38 EPU-4X62 EPU-5070	Basic Configuration 0002	0xC80
EPU-3311 EPU-3312	Basic Configuration 0003	0x1C80

Once changed, reboot the board for the configuration to take effect.

Note 1. These products do not use the driver address setting and default to Basic Configuration 0001 0xCA0.

Appendix C. Sample Code

The file *versaAPI_example.tgz* included in the Linux release package contains example VersaAPI code that can be used as a starting point for code development and as an acceptance test for your VersaAPI installation.

To use the example:

1. Create a temporary directory and copy the versaAPI_example.tgz file into it.
2. Expand the file (tar xvzf versaAPI_example.tgz). The compressed file contains three files:
    a. README - Text file explaining the example.
    b. Makefile - Linux compatible make file.
    c. versaAPI_sample.c - C source code for the example.
3. Copy the VersaAPI include file VL_OSALib.h into the current directory.
    a. cp ../../VL_OSALib.h .
4. Make the example using the Linux 'make' command:
    a. For a EPU-331X board use: make clean; make epu-331x
    b. For a non EPU-331X board use: make clean; make
5. Run the example with the desired command line options from the Linux prompt: ./versaAPI_sample

The code is an example of how to:

- Open/close the VersaAPI library.
- Retrieve the release version of the VersaAPI library.
- Access onboard DIO.
- Access EPMe-A1/A2 DIO.
- Use of the 8254 timer code including how to write and register a callback function for when the timer expires.

Other examples are included in the examples directory for our various boards. We will continue to add example applications as they are developed.

Appendix D. MPEu-C1 Details Regarding Timing

The default supported clocking rates for the MPEu-C1 (LPC54616 MCAN interface) consist of four main parts:

Master Divider (master clock rate divider) - What the master clock is divided by and is the clocking rate of the MCAN interface. This divider is common to both the arbitration and data bit clocks.

preDividerf (MCAN clock divider) - The master clock is divided by this rate and it is the clock rate for the bit time quanta. The value programmed into the NBTP and DBTP registers is the value-1. The arbitration and data clocks have their own dividers for this.

Notes:

- Arbitration and Data bits have their own Seg values with different upper limits.
- Values put into NBTP and DBPT registers are (calculated value – 1).
- Upper limit values for Seg1 and Seg2 differ.

Seg1 - Number of quanta clocks for the propagation for phase1 part of a bit.

Seg2 - Number of quanta clocks for phase2 part of a bit.

Calculate the number of quanta, which must be an integer:
mcanClkRate = masterClock / (masterDivider * (mcanDivider + 1));
numberOfQuanta = mcanClkRate / bitRate;

Notes:

- The value 0.7 (70%) is where you want the data sampled in the CAN bit. Some implementations use 0.8 (80%).
- We typically used 3 for the Arbitration jump (sync) bit value.

```
seg1 = int( (numberOfQuanta - 3) * 0.7 );
seg2 = (numberOfQuanta - 3) - seg1;
```

```
realBitRateCalc = ((180000000 / masterDivider) / (preDivider + 1)) / (seg1 + seg2 + 3);
```

Table 39: MPEu-C1 Default Supported Arbitration/Data Bit Rates

Arbitration Rate	Data Rate	Master Divider	Arbitration preDivider	Arbitration jump	Arbitration Seg1	Arbitration Seg2	Data preDivider	Data jump	Data Seg1	Data Seg2
10000	10000	24	24	3	20	7	24	3	20	7
10000	20000	24	24	3	20	7	14	3	16	6
10000	50000	24	24	3	20	7	4	3	20	7
10000	100000	24	24	3	20	7	2	3	16	6
10000	125000	24	24	3	20	7	2	3	12	5
20000	20000	20	14	3	20	7	14	3	20	7
20000	50000	20	14	3	20	7	5	3	20	7
20000	100000	20	14	3	20	7	2	3	20	7
20000	125000	15	19	3	20	7	2	3	21	8
20000	250000	12	24	3	20	7	2	3	12	5
50000	50000	18	7	3	16	6	7	3	16	6
50000	100000	18	7	3	16	6	3	3	16	6
50000	125000	18	7	3	16	6	3	3	12	5
50000	250000	18	7	3	16	6	1	3	12	5
50000	500000	18	7	3	16	6	0	3	12	5
100000	100000	5	11	3	20	7	11	3	20	7
100000	125000	5	11	3	20	7	8	3	21	8
100000	250000	4	14	3	20	7	8	3	12	5
100000	500000	4	14	3	20	7	4	3	11	4
100000	800000	3	19	3	20	7	2	3	16	6
100000	1000000	3	19	3	20	7	2	3	12	5
125000	125000	4	17	3	12	5	17	3	12	5
125000	250000	4	17	3	12	5	8	3	12	5
125000	500000	4	17	3	12	5	4	3	11	4
125000	800000	3	19	3	20	7	2	3	16	6
125000	1000000	3	19	3	20	7	2	3	12	5
250000	250000	3	11	3	12	5	11	3	12	5
250000	500000	3	11	3	12	5	11	3	5	2
250000	800000	3	11	3	12	5	2	3	16	6
250000	1000000	5	8	4	9	4	1	3	11	4
250000	2000000	5	8	4	9	4	0	4	11	4
500000	500000	5	3	3	11	4	3	3	11	4
500000	800000	5	3	3	11	4	2	3	9	3
500000	1000000	9	1	3	12	5	0	3	12	5
500000	2000000	5	3	3	11	4	0	3	11	4
500000	3000000	4	4	3	11	4	0	3	9	3
800000	800000	5	2	3	9	3	2	3	9	3
800000	1000000	5	2	3	9	3	1	3	11	4
800000	2000000	5	2	3	9	3	0	3	11	4
800000	3000000	3	2	3	16	6	0	3	12	5
1000000	1000000	5	1	3	11	4	1	3	11	4
1000000	2000000	5	1	3	11	4	0	3	11	4
1000000	3000000	3	2	3	12	5	0	3	12	5
1000000	5000000	3	2	3	12	5	0	3	7	2