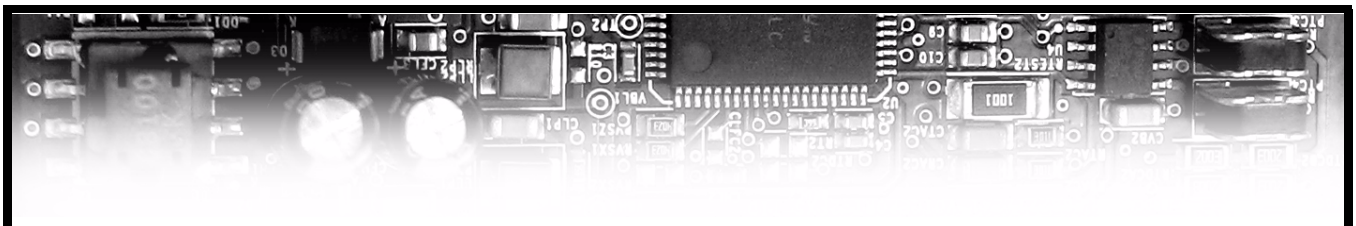


VCP2 Package

QuickStart Guide

Part Number:	**VCP2**
Revision Number:	**A1**
Issue Date:	**January 2008**

ZARLINK SEMICONDUCTOR

FEATURING Legerity® VOICE SOLUTIONS

TABLE OF CONTENTS

CHAPTER

1 INTRODUCTION

1.1 OVERVIEW

The Voice Control Processor (VCP2) Package is a collection of software, tools, code examples, and documents that use the Zarlink VCP2 chipsets to promote rapid application development. With the VCP2 Package, little or no knowledge of the low-level control of Zarlink ICs is needed to fully utilize the chipset. The VCP2 is designed to streamline POTS line card requirements, simplify implementation and reduce customers' time to market. The VCP2 system reduces the complexity and real-time constraints associated with call control. Detecting DTMF tones, sending caller ID messages, and controlling line voltage characteristics are examples of the kind of call control tasks the VCP2 can perform with minimal host interaction.

This Quick Start Guide provides an overview of the VCP2 system and how the Package is used to implement a control system for the VCP2.

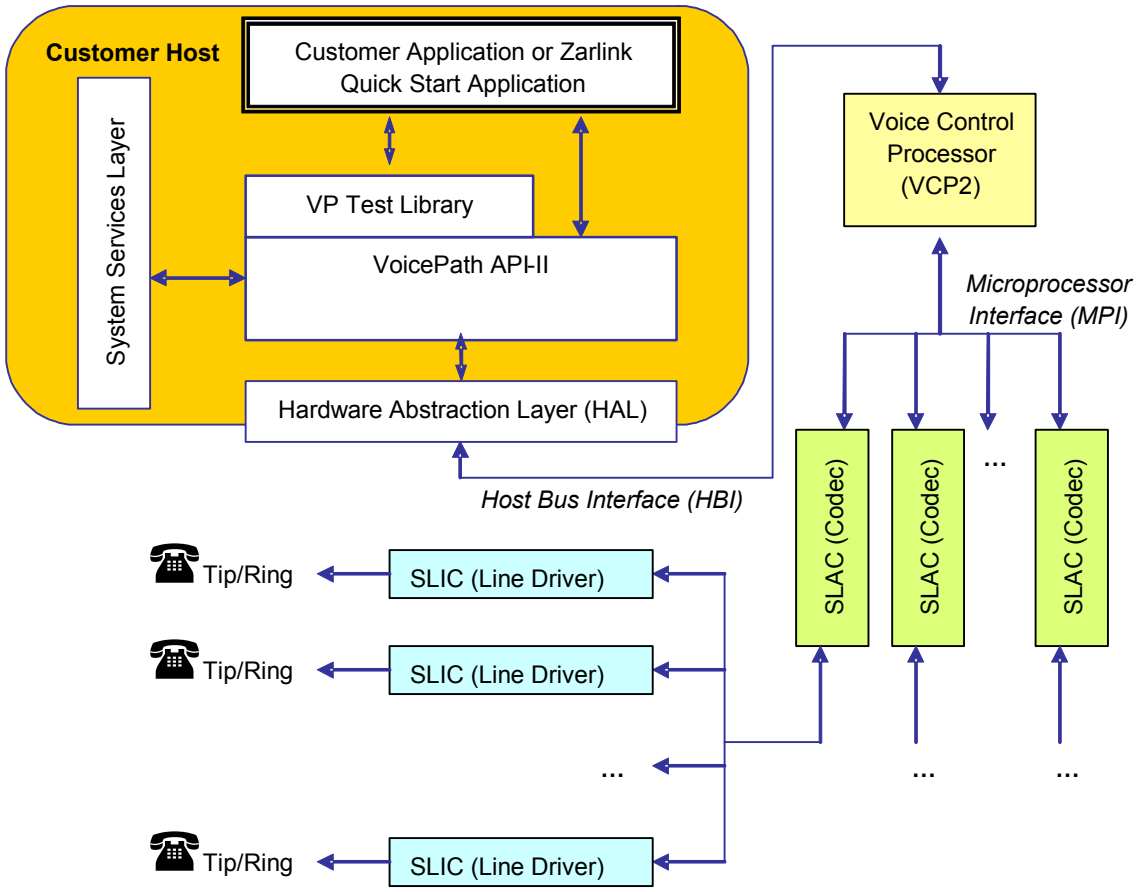
Note:

In this document, the VoicePath™ API-II may simply be referred to as VP-API-II.

1.2 VCP2 SOFTWARE DESIGN FLOW

The following diagram introduces the prominent elements of the VCP2 system architecture. It is recommended the user read the VP-API-II Reference Guide introduction for more precise details on these elements.

Figure 1–1 System Diagram



1.3 VCP2 PACKAGE COMPONENTS AT A GLANCE

The VCP2 Package contains tools, code, documentation, and examples for using the VCP2 chipset. The following is a list of components distributed in the VCP2 Package.

- VP-API-II (source and documentation)
- VCP2 Test Library (source and documentation) -- optional with purchase of BT, AT, or ATP packages.
- VCP2 Software Data Sheet
- VCP2 Firmware Image
- VCP1 to VCP2 Conversion Guide
- Profile Wizard
- Signal Integrity Tools
- Boot Image Tools
- Quick Start Application
- HAL & SSL Examples

To find more about these items and where they (and their documentation) can be located, refer to Table 1 below. The pathnames found in the table use "xxx" to represent version numbers and product strings that are subject to change.

Table 1–1 Common Components List

Component	Description	Documentation
VP-API-II	This C-language library provides a common, consistent interface to Zarlink voice termination devices. The API, which is executed on a host processor, simplifies the task of communicating with the VCP2. ***Note:*** *The Hardware Abstraction Layer (HAL) and System Services Layer (SSL) are integral to the VP-API. These modules must be tailored to meet the specific underlying platform.* **Location**: /vcp2_xxx/api_lib	VP-API-II Reference Guide -the primary reference to consult when developing VCP2 application software. ("/vcp2_xxx/documents")
VCP2 Firmware Image	This firmware image must be loaded on the VCP2 through the VpBootLoad VP-API function each time the VCP2 is powered-up. The image is specific to the type of VCP2 package purchased. The firmware image is also referred to as the boot image. **Location**: /vcp2_xxx /firmware/vcp2/ vcp2_xxx.hbi. Compressed images are denoted by appending a number indicating the compression window size, followed by a z. For instance, "vcp2_123.hbi8z"	Release notes are included in the same directory. Refer to the VP-API Reference Guide "VpBootLoad" section for more information about boot images.
Profile Wizard	Profile Wizard is a graphical Windows™ application to help developers create configuration profiles used by the VCP2. Profiles are loaded during normal operation to define regional and design specific attributes such as call progress tone frequencies, the resistance threshold for detecting on-hook events, and MPI bus clock speed. **Location**: /vcp2_xxx/tools/pw_xxx.exe The Profile Wizard provided in the VCP2 package release is the latest version at the time of release. Profile Wizard, however, will be updated separately from the VCP2 package. The latest version of Profile Wizard can be obtained from the Zarlink web site at http://www.zarlink.com.	Profile Wizard on-line help (internal to application)

Table 1–1 Common Components List

Component	Description	Documentation
Signal Integrity Tools	Tools to test the HBI and MPI connections. HBI Test - Use this tool to ensure correct operation of the HAL functions for low-level communication between the host and the VCP2. MPI Test - Use this tool to test MPI signal integrity between the VCP2 and the SLAC devices. **Location**: /vcp2_xxx/tools/	Quick Start Guide Chapter 3, on page 7 Chapter 4, on page 9
Boot Image Tools	The VCP2 Package includes Windows command-line utilities to reformat the boot image. They can be found in the "/tools" directory of the product package. *ldr2c.exe* - converts firmware images into a C declaration format that can be statically compiled into the host software application. *deflater.exe* - compresses the VCP2 firmware image into a smaller image which can be decompressed at runtime by the VpBootLoad() function. **Location**: /vcp2_xxx/tools/	Quick Start Guide Chapter 6, on page 15
Quick Start Application	This source code may be compiled to create an application that demonstrates functional aspects of the VP-API-II. **Location**: /vcp2_xxx/apps/qs_xxx	Quick Start Guide Chapter 5, on page 11
HAL & SSL Examples	These C language samples of Hardware Abstraction Layer and System Services Layer implementations provide examples of the functions the user must implement for their architecture. Extensively commented. **Location**: /vcp2_xxx/arch ***Note:*** *There are several samples of the HAL and System Services Layer. User should choose the one that is most appropriate for their application when starting development of a VCP2 application.*	VP-API-II Software Reference Guide

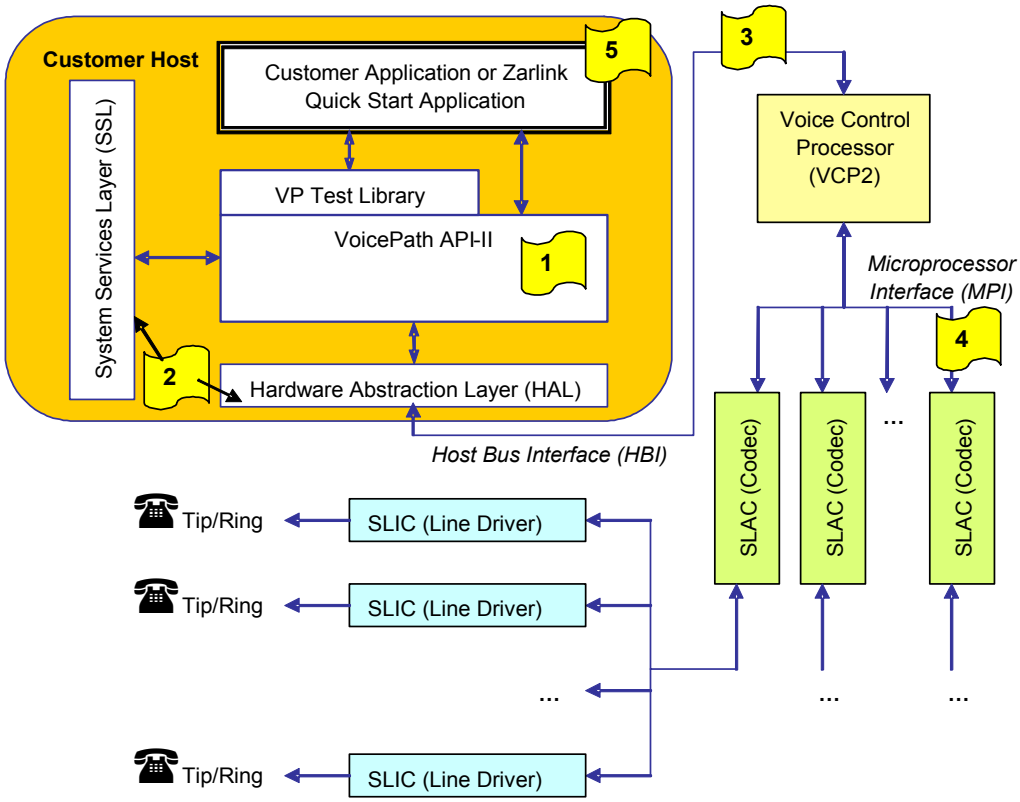
CHAPTER

2 FIRST STEPS

There is a natural sequence to early VCP2 system development to ensure quick and proper system operation. The first steps in bringing up a new VCP2 design are shown in the following list and diagram.

1. Configure the API. The API contains several pre-processor options. They need to be configured in the API header file "vp_api_cfg.h" located in the "includes" directory of the VP-API source tree. Comments in this file provide directions on how to select the appropriate chipset, default options, and device context size.
2. Write/Modify the SSL and HAL modules. These thin layers provide functions allowing the API to enforce mutual exclusion and to communicate with the VCP2 through the HBI bus. Examples of implementations of system service and hardware abstraction layers are available in the "/vcp2_xxx/arch" directory of the product package.
3. Test the HBI. Use the provided HBI Memory Test application to verify that the VCP2's memory space is accessible to the host and can be read/written at the speed that the host is expecting. See Chapter 3, on page 7 of this guide.
4. Test the MPI bus. Use the provided MPI Test to ensure MPI bus signal integrity between the VCP2 and SLACs. See Chapter 4, on page 9 of this guide.
5. Run the Quick Start application. Configure and run the Quick Start application to ensure the end to end operation of the system from the host to Tip/Ring. See Chapter 5, on page 11 of this guide.
6. Write the end user application. The MPI, HBI, SSL, and HAL components are now functional, greatly simplifying the control application troubleshooting process. Refer to the VP-API-II Reference Guide for information on the API interface used by the host program to control the VCP2.

Figure 2–1 Steps To Take



CHAPTER

3 HBI MEMORY TEST

3.1 OVERVIEW

Before booting the VCP2 for the first time, a user should execute the HBI Memory Test located in located in directory "*install_dir*/sdk/tools/hbi_mem_test". This program writes data to the VCP2 over the HBI bus using the Hardware Abstraction Layer functions and then checks to see if the data can be read back correctly.

3.2 USER MODIFICATIONS

The user may need to make customizations to the HBI Memory Test Application prior to execution. The next two subsections will explain such customizations.

3.2.1 Test Options

The type of testing performed can be controlled in "vtd_mem_test.c" by use of the preprocessor options listed below.

Table 3–1 Debug Options

Debug Option	Defined by Default	Purpose (when option is defined)
HBI_HAL_DEBUG	Yes	Enables display of progress and debug information.
EXIT_ON_HW_VERSION_REG_FAIL	No	Exits upon failure to read the VCP2 Hardware Version Register.
NEED_MORE_DEBUG	No	Prints more information regarding the address being read, the value written into that address and the value that is being read back. Note that this is used during the small block testing.
TEST_MULTI_WORD_RD_WR	Yes	Performs the memory multi-word read write test.

3.2.2 Board Initialization

The file "brd_init.c" contains the function InitializeBoard() which is used to perform any specific board initialization that may be needed. The customer needs to change this function based on their hardware platform. In many cases, InitializeBoard() may be left as a stub (all content commented out). But other systems may need to perform some initialization to set up components essential to HBI communication.

The function as shipped is designed to work with a Zarlink demonstration platform. In this custom implementation, the following tasks are performed: configure the on-board CPLD, assert the VCP2's reset signal, set the VCP2's HBI configuration (CONFx) pins, and then deassert the VCP2's reset signal.

3.3 BASIC PROGRAM OPERATION

Before the memory testing is performed, the program first checks to see if it can identify the VCP2 hardware silicon version by reading the hardware silicon version register. It then reads and displays the PCLK register, which gives the PCM clock speed (see Chipset User's Guide for the system's SLAC). Then the module tests the HAL functions with some a memory test. The test concludes with

a message indicating success if it recognizes the silicon revision version and the memory test passes. A sample output for the HBI Memory test using a Zarlink demonstration board is shown in Appendix A.

3.4 TROUBLESHOOTING

The failure of a memory test can occur due to:

- PCLK or FS not being present at time of VCP2 boot.
- Problems in the HAL implementation software
- Problems in the hardware
- Configuration error in HBI test software

Use the debug options listed under the User Modifications section to help in troubleshooting.

CHAPTER

4 MPI TEST

4.1 OVERVIEW

The MPI Test located in directory "*install_dir*/sdk/tools/vcp2_mpi_test" is used to help verify correct operation and signal integrity of the MPI bus between the VCP2 and the SLACs.

Note:

The MPI Test does not replace good layout practices and full signal integrity characterization using high performance scopes and other lab equipment. It simply performs several read/write transactions between the VCP and the SLAC looking for errors, and does not guarantee MPI signals meet the data sheet requirements.

The test is composed of a host application (C-language source provided in the VCP2 Package) and a special VCP2 firmware image that performs a read/write test to all SLAC devices detected by the VCP2.

Remember that the HBI must be free from errors for the MPI Test to function reliably. See Chapter 4, on page 9. First Steps for more information about the recommended design process.

4.2 USER MODIFICATIONS

There are several platform specific HBI test configurations that have to be done by the user before the MPI Test application can be executed. The files that need to be modified are described below.

4.2.1 Define Appropriate VoicePath Device Profile

Default profiles are provided in the MPI Test application in the file profiles8_790.c. The device profile should be altered to reflect the user's platform configuration. Use Profile Wizard located in the tools directory of the VCP2 Package to create an appropriate device profile. For more information on profiles, refer to the Profiles Section of the VP-API-II User's Guide.

4.2.2 Modify brd_init.c

The file "brd_init.c" contains the function InitializeBoard() which is used to perform any specific board initialization that may be needed. The customer needs to change this function based on their hardware platform. In many cases, InitializeBoard() may be left as a stub (all content commented out). But other systems may need to perform some initialization to set up components essential to HBI communication.

The function as shipped is designed to work with a Zarlink demonstration platform. In this custom implementation, the following tasks are performed: configure the on-board CPLD, assert the VCP2's reset signal, set the VCP2's HBI configuration (CONFx) pins, and then deassert the VCP2's reset signal.

4.2.3 Configure Precompiler Options

4.2.3.1 app_des.h

This file contains preprocessor options describing device and board configuration specific to the application.

4.2.3.2 mpi_test_vcp.c

The preprocessor macro term_printf may need to be redefined to work with an architecture's display function. To poll the VCP2 instead of working through an interrupt service routine, define POLL_VCP_EVENTS. To exit the program upon reaching an error, define EXIT_ON_ERROR. Otherwise, the program will never terminate.

4.3 USING THE PROGRAM

The MPI Test host application will boot and initialize the VCP2 with the MPI Test firmware image. The program will display the message Device initialization complete. Detected SLACs: xxx where xxx is a bit-field representing the chip-selects of SLACs found attached to the VCP2. The user should check this value against the expected number of SLACs. For example, 0000 0000 0000 0111 indicates that the VCP2 initialized 3 SLACs (on chip-selects 0,1, and 2).

After initialization is complete, the test will listen for status messages from the VCP2 passed in the form of VP_DEV_EVID_STEST_CMP (self-test complete) events. A self-test complete event's eventData field will be equal to 0 if no MPI transaction has failed. The test will run until there is an MPI error or until the board is reset. It takes approximately 20 ms after VCP2 initialization to run a full MPI test. Wait for at least two self-test complete events to ensure a complete MPI test is run (while the self-test events contain information about the MPI tests, the frequency of the events is timer-driven and does not directly reflect the number of MPI tests run). It is suggested that the MPI test run for many thousands of iterations (100,000 iterations takes ~30 seconds, if successful). In the event of an MPI error, a VP_DEV_EVID_STEST_CMP event will be returned immediately with failure information. A sample output for the MPI test using a Zarlink demonstration board is shown in Appendix A.

Note:

Do not use the MPI Test image for regular operation of the VCP2. The only API functions that are supported by this firmware image are system configuration functions (VpMakeDeviceObject, VpMakeLineObject, VpGetDeviceInfo, VpGetLineInfo, VpFreeLineCtx, VpMakeDeviceCtx, VpMakeLineCtx), and the functions VpBootLoad, VpInitDevice, VpSetOption, VpGetEvent, and VpGetResults.

4.4 BASIC PROGRAM OPERATION

The sequence of events the MPI Test host application performs is:

1. Make the device object and context (explained in the VP-API-II Reference Guide).
2. Call VpBootLoad with the MPI Test VCP2 image, passing a valid device profile, with all other profiles specified during VpBootLoad set to VP_PTABLE_NULL. It is important that all other profiles are null; otherwise the test will not be performed. Poll for the expected VP_DEV_EVID_BOOT_CMP event and its associated results using the VpGetEvent and VpGetResults API commands. After the VCP2 has been booted, the MPI will automatically begin testing.

Note:

The VCP2 will not boot correctly unless PCLK and FS are both connected at time of boot.

3. Create line objects and contexts. Refer to the VP-API-II Reference Guide for more information.
4. Issue a VpSetOption command for the VP_OPTION_ID_EVENT_MASK option with the VP_DEV_EVID_STEST_CMP event type unmasked.
5. Poll the VpGetEvent command for events. A VP_DEV_EVID_STEST_CMP event should be received approximately once every second after boot load until an MPI error is found. If the MPI bus is operating properly, these "self test complete" events should contain 0 in the eventData field. Otherwise, the event will contain the following pieces of information:

   a) The channelId field will have the chip-select of the SLAC which experienced MPI problems. For 790 chipsets, the received channelId represents the chip-select of the SLAC in question.

   b) The eventData field will have the expected MPI data. For 790 chipsets, the expected data is stored in the upper byte, and the data received from the SLAC in the lower byte.

Observed MPI errors may be due to an incorrect clock setting, a bad physical connection, or cross talk on the MPI bus.

CHAPTER

5 QUICKSTART APPLICATION

The Quick Start Application located in directory "*install_dir*/sdk/apps/vcp2_example" demonstrates some of the functionality of the VP-API-II. The purpose of this application is to ensure that the major system components (VCP2, SLAC, SLIC, HAL, SSL, and API) are configured properly without the user having to write custom application code. The Quick Start application provides an end-to-end sanity check and illustrates the functionality of the platform.

5.1 OVERVIEW

The Quick Start application uses a telephone hooked into the VTD for input. Its behavior is based on a combination of precompiler macros, runtime options controlled by DTMF signals, and a state machine triggered by hook events. The application provides output to the phone in forms such as tone, ringing, and metering, and will display debug and help information on the host via a serial port.

The Quick Start application implements the following high-level state machine.

Table 5–1 QuickStart State Machine

State Hook	Transition	Action
State 1	Reset on hook	OHT state
State 2	First off hook	Dial tone
State 3	First on hook	Ringing
State 4	Second off hook	Morse code message "CQ...."
State 5	Second on hook	Go back state 1

5.2 USER MODIFICATIONS

There are several platform specific HBI test configurations that have to be done by the user before the MPI Test application can be executed. The files that need to be modified are described below. This section assumes the user has completed configuring the API and written the SSL and HAL modules. If not, consult the "First Steps" section of this guide.

5.2.1 Translate the Boot Image

The Quick Start application expects a boot image to be stored in the file boot_image_xxx.h. Use the boot image converter ldr2c.exe located in the tools directory of the VCP2 Package to convert the VCP2 firmware image (located in the /vcp2_xxx /firmware/vcp2 directory of the VCP2 Package) into a C-language format. This tool is explained in Chapter 6, on page 15.

5.2.2 Define Appropriate VoicePath Profiles

Default profiles are included in the Quick Start Application source code inside profiles8_xxx.c. The device profile should be altered to reflect the user's platform configuration. Use Profile Wizard, located in the tools directory of the VCP2 Package to create an appropriate device profile. Custom AC and DC profiles may also be desired but not required for this quick-start application. For more information on profiles, refer to the Profiles section of the VP-API Reference Guide.

5.2.3 Modify brd_init.c

The file "brd_init.c" contains the function InitializeBoard() which is used to perform any specific board initialization that may be needed. The customer needs to change this function based on their hardware platform.

Note:

The implementation of functions in brd_init.c will be the same as those functions from Section 4.2.2. See Section 4.2.2 for more details.

5.2.4 Configure app_des.h

This file contains declarations that are specific to the platform being used. Review the section commented as "Compile Time Options" and configure as needed. Some of the variable definitions that might have to be changed are MAX_DEVICES_ON_BOARD, MAX_LINES_ON_BOARD and MAX_DTMF_DETECTION_LINES.

5.2.5 Configure framer_dummy.c (optional)

This file contains a dummy PCM framer interface. Since this is a stub implementation, all the functions in this file currently perform no action other than printing a debug statement. They may be replaced with the appropriate implementation if a PCM framer is present and desired. This step may only be required if the clocks provided to the SLAC are derived from a framer or other external source.

5.3 BASIC PROGRAM OPERATION

The program first performs any platform initialization that may be necessary. It then initializes and configures the framer. The Quick Start application then creates and initializes the VCP2 device context and boots the VCP2 with the VpBootLoad function. Line contexts are created and the VCP2 is initialized with the VpInitDevice function. Possible/expected events are then unmasked with VpSetOption since most events are masked by default. The Quick Start application then performs further initializations including initializing ringing parameters, and configuring the PCM highway.

The application performs actions such as enabling DTMF detection, getting the current line state, and placing the line into the on-hook transmission state (OHT) depending on the events received from the VCP2. Events from the VCP2 are handled by a function called the VtdEventHandler(). This function can be called from the interrupt context or from a routine that is polling for events. In either case this function is passed a pointer to a structure that was filled by the VpGetEvent() function.

A sample output for the QuickStart Application using a Zarlink demonstration board is shown in Appendix A.

5.4 USING THE PROGRAM

Perform the following exercise to verify that the Quick Start application is running correctly:

1. Connect a standard POTS telephone to either line 1 or line 2 of the VTD.
2. Lift the handset off-hook and listen for a dial tone.
3. Hang up the telephone and verify that the phone starts ringing.
4. Take the handset off-hook again and listen for a morse code.
5. Hang up the telephone and verify that it does not ring.

5.5 HELP OPTIONS

The Help options can be displayed by attaching a phone to line 0, going off-hook, and pressing the "*" button on the telephone keypad. Five options are available to the user, and each of them is explained below:

Table 5–2 QuickStart Help Options

Option	Description
*	Displays a list of available options.
00	Displays the results from the VpGetOption function. Currently only two options are supported. Access these by keying in the following codes: VP_DEVICE_OPTION_ID_PULSE: **00 0*** VP_OPTION_ID_EVENT_MASK: **00 10***
20	Starts metering. The command to start metering is: **20 X*** where X is the desired number of metering pulses.

CHAPTER

6 BOOT IMAGE TOOLS

6.1 OVERVIEW

Two executables for converting the VCP2 boot image are included in the VCP2 Package. The first (deFlater.exe) compresses a boot image, reducing the amount of memory required to store it. The VP-API can then decompress the image as it is loading it into the VCP2. The second tool (ldr2c.exe) will take a firmware image and generate a header file containing C-language code that contains the boot image as an array. This is provided to allow simple inclusion of the boot image within C applications.

Both programs are command-line utilities for Microsoft Windows and can be found in the "/VCP2_xxx/tools" directory of the VCP2 Package.

6.2 LDR2C

This utility converts a binary VCP2 firmware image into C declaration format for use and inclusion as a ".h" file. This allows the firmware image to be statically compiled into the host application. The resulting C code declares an unsigned char array initialized with the firmware image data.

Usage: ldr2c image_file c_file

image_file - the binary firmware image file (*.hbi) to be converted

c_file - the C source code file (or header file) in which to place the results

Example Output:

```
>ldr2c VCP2_v1.86_rc_14.hbi boot_image.h
Opening file 'VCP2_v1.86_rc_14.hbi' for input... done.
Opening file 'boot_image.h' for output... done.
Creating C file from boot stream file... done.
```

6.3 DEFLATER

This program can be used to compress a VCP2 firmware image. The compressed image can be decompressed on-the-fly by VpBootLoad() before booting the VCP2. Eight different levels of compression (8 to 15) are available, with tradeoffs between resulting image size, speed of decompression during the bootload process, and amount of memory required in the host to decompress.

Compressed images take up approximately half the memory of uncompressed images. This is particularly useful when code memory is limited and the firmware image is to be compiled into the host application (sending the output of this tool to ldr2c will give you a header file version of the compressed image). There is a memory overhead during decompression defined by sizeof(VpScratchMemType) which is typically between 8-10k, depending on what options are set. To find out more about compressed images, refer to the VpBootLoad function description in the VP-API-II Reference Guide and the comments in vp_api_cfg.h.

Usage: deflater [-winbits] infile [outfile]

winbits - level of compression, from 8 (worst) to 15 (best); default is 8

infile - file to be compressed

outfile - file for compressed output

Example Output:

```
>deflater -8 VCP2_v1.86_rc_14.hbi VCP2_v1.86_rc_14.hbiz
```

```
Zarlink Deflater v1.1

       Deflating: VCP2_v1.86_rc_14.hbi -> VCP2_v1.86_rc_14.hbiz
 Dictionary size: 256 bytes
 Input file size: 163 kB
Output file size: 93 kB (56%)
Adler32 checksum: 187F2FB8h
   Verifying: inflated to 163 kB
```

CHAPTER

7 TECHNICAL SUPPORT

Go to http://www.zarlink.com to get the latest product information and documentation.

APPENDIX

A SAMPLE OUTPUTS

A.1 SAMPLE OUPUT FOR HBI MEMORY TEST

The following output was created using the UVB. The output shown indicates the HBI bus is properly configured.

```
Configuring DIN connector in HBI interface mode...
Holding the DIN connector in reset
Setting HBI clock
setting HBI config pins
Releasing the DIN connector reset
Board Setup complete.
Initializing CPLD and PINCONFIG register...
Selecting Page 255
Verifying HWREVCODE register...
Setting Page 255 base register: 0x3fe0
HBI-Access:Add:0xffff, Read: 0x   b

*********************************************
Revision ABA VCP2 or VPP silicon found.

*********************************************
PCLKSEL Register = 0x03ff
Please make sure this register setting is correct based on the
clock signal provided to the VCP2/VPP device.
NOTE: By default this register is configured to autodetect
PCLK freq given FS.

Starting Memory Test...
Testing DM FIXED >>>
-----------------------------------------------------------------
<<< Testing DM FIXED Done
Testing DM PAGE 0 >>>
-----------------------------------------------------------------
<<< Testing DM PAGE 0 Done
Testing DM PAGE 1 >>>
-----------------------------------------------------------------
<<< Testing DM PAGE 1 Done
Testing PM PAGE 0 >>>
----------------
<<< Testing PM PAGE 0 Done
Testing PM PAGE 1 >>>
----------------
<<< Testing PM PAGE 1 Done
Testing PM PAGE 2 >>>
----------------
<<< Testing PM PAGE 2 Done
Testing PM PAGE 3 >>>
----------------
<<< Testing PM PAGE 3 Done
Starting Memory multi word (Access size = 16) HBI access verification...
Testing DM FIXED >>>
-----------------------------------------------------------------
<<< Testing DM FIXED Done
Testing DM PAGE 0 >>>
-----------------------------------------------------------------
<<< Testing DM PAGE 0 Done
Testing DM PAGE 1 >>>
-----------------------------------------------------------------
<<< Testing DM PAGE 1 Done
Testing PM PAGE 0 >>>
```

```
----------------
<<< Testing PM PAGE 0 Done
Testing PM PAGE 1 >>>
----------------
<<< Testing PM PAGE 1 Done
Testing PM PAGE 2 >>>
----------------
<<< Testing PM PAGE 2 Done
Testing PM PAGE 3 >>>
----------------
<<< Testing PM PAGE 3 Done

SUCCESS:: Memory test passed
```

A.2 SAMPLE OUPUT FOR MPI TEST

The following output from the MPI Test shows a system with 1 SLAC connected to the VCP2. After a few seconds of testing, the SLAC is removed from the VCP2 and an MPI failure is generated.

```
mpi_test
Switching on the battery for the SLICs...
Configuring DIN connector in HBI interface mode...
Holding the DIN connector in reset
Setting HBI clock; 8 MHz
setting HBI config pins
Releasing the DIN connector reset
Board Setup complete.


Welcome to the Zarlink VP-API MPI Test application.

Creating Device Object...done
Downloading VCP2 boot image...done
Received event: VP_EVID_BOOT_CMP
Reading Checksum...VpEvent Cat:Id = VP_EVCAT_RESPONSE:VP_DEV_EVID_BOOT_CMP
VpEventType.channelId = 0x0
VpEventType.deviceId = 0x0
VpEventType.eventData = 0x0
VpEventType.parmHandle = 0x0
VpEventType.hasResults = 0x1
done. Checksum Data:
VpChkSumType.loadChkSum = 0xc91434dc
VpChkSumType.vInfo.vtdRevCode = 0xb
VpChkSumType.vInfo.swProductId = 0x4
VpChkSumType.vInfo.swVerMajor = 0x1
VpChkSumType.vInfo.swVerMinor = 0x88
creating Line Object 0, TermType = VP_TERM_FXS_TO_TL...done
creating Line Object 1, TermType = VP_TERM_FXS_TO_TL...done
creating Line Object 2, TermType = VP_TERM_FXS_TO_TL...done
creating Line Object 3, TermType = VP_TERM_FXS_TO_TL...done
Initializing VCP2...done
Setting default options for VCP2(All Lines)...
Received event: VP_DEV_EVID_DEV_INIT_CMP(0)
Device initialization complete Event Data: 0x1
Setting up events masks...
done
No mpi errors...
No mpi errors...
No mpi errors...
No mpi errors...
No mpi errors...
No mpi errors...
No mpi errors...
No mpi errors...
No mpi errors...
MPI problem detected on SLAC #1: expected 0xA8, got 0xFF
ERROR: Unknown event or Error while handling the event; Printing the event...VpE
vent Cat:Id = VP_EVCAT_TEST:VP_DEV_EVID_STEST_CMP
VpEventType.channelId = 0x1
```

```
VpEventType.deviceId = 0x0
VpEventType.eventData = 0xa8ff
VpEventType.parmHandle = 0x0
VpEventType.hasResults = 0x0
ERROR: VtdEventHandler = VP_STATUS_ERR_VERIFY.

ERROR: Unknown event or Error while handling the event; Printing the event...VpE
vent Cat:Id = VP_EVCAT_FAULT:VP_DEV_EVID_CLK_FLT
VpEventType.channelId = 0x0
VpEventType.deviceId = 0x0
VpEventType.eventData = 0x1
VpEventType.parmHandle = 0x0
VpEventType.hasResults = 0x0
ERROR: VtdEventHandler = VP_STATUS_FAILURE.
```

A.3 SAMPLE OUPUT FOR QUICKSTART APPLICATION

This output shows the QuickStart application initializing a system with four lines and interacting with a user. This output resulted from a user picking up a phone on line 0 and pressing the "*" key.

```
Switching on the battery for the SLICs...
Configuring DIN connector in HBI interface mode...
Holding the DIN connector in reset
Setting HBI clock; 8 MHz
setting HBI config pins
Releasing the DIN connector reset
Board Setup complete.

Welcome to the Zarlink VP-API quick start demo application.

Initializing Framer...
Creating Device Object...done
Downloading VCP2 boot image...done
Received event: VP_EVID_BOOT_CMP
Reading Checksum...VpEvent Cat:Id = VP_EVCAT_RESPONSE:VP_DEV_EVID_BOOT_CMP
VpEventType.channelId = 0x0
VpEventType.deviceId = 0x0
VpEventType.eventData = 0x0
VpEventType.parmHandle = 0x0
VpEventType.hasResults = 0x1
done. Checksum Data:
VpChkSumType.loadChkSum = 0xc30f0b5c
VpChkSumType.vInfo.vtdRevCode = 0xb
VpChkSumType.vInfo.swProductId = 0x4
VpChkSumType.vInfo.swVerMajor = 0x1
VpChkSumType.vInfo.swVerMinor = 0x88
creating Line Object 0, TermType = VP_TERM_FXS_TO_TL...done
creating Line Object 1, TermType = VP_TERM_FXS_TO_TL...done
creating Line Object 2, TermType = VP_TERM_FXS_TO_TL...done
creating Line Object 3, TermType = VP_TERM_FXS_TO_TL...done
Initializing CID Profile table
Initializing VCP2...done
Setting default options for VCP2(All Lines)...
Received event: VP_DEV_EVID_DEV_INIT_CMP(0)
Device initialization complete Event Data: 0x1
Setting up events masks...
Configuring line 1 for different DC feed, Ringing Profile...done
Line 0 Ring Init, LCAS State->Normal, TimeSlot->0, CODEC->Mu-Law, DTMF->ON, DTMF
 Status=0x00000001, DTMF Remaing Resources=15, Line State: DISCONNECT -> OHT
Line 1 Ring Init, LCAS State->Normal, TimeSlot->2, CODEC->Mu-Law, DTMF->ON, DTMF
 Status=0x00000003, DTMF Remaing Resources=14,
DTMF->OFF, DTMF Status=0x00000001, DTMF Remaing Resources=15,
Line State: DISCONNECT -> OHT
Line 2 Ring Init, LCAS State->Normal, TimeSlot->4, CODEC->Mu-Law, DTMF->ON, DTMF
 Status=0x00000005, DTMF Remaing Resources=14, Line State: DISCONNECT -> OHT
Line 3 Ring Init, LCAS State->Normal, TimeSlot->6, CODEC->Mu-Law, DTMF->ON, DTMF
 Status=0x0000000d, DTMF Remaing Resources=13, Line State: DISCONNECT -> OHT
Issuing Get loop condition command...done
```

```
LoopCond: VpLoopCondResultsType.rloop = 32767
LoopCond: VpLoopCondResultsType.ilg = 16
LoopCond: VpLoopCondResultsType.imt = -8
LoopCond: VpLoopCondResultsType.vsab = 9908
LoopCond: VpLoopCondResultsType.vbat1 = -9634
LoopCond: VpLoopCondResultsType.vbat2 = -29047
LoopCond: VpLoopCondResultsType.vbat3 = 463
LoopCond: VpLoopCondResultsType.mspl = 0
LoopCond: VpLoopCondResultsType.selectedBat = VP_BATTERY_2
LoopCond: VpLoopCondResultsType.dcFeedReg = VP_DF_RES_FEED_REG
Received event: VP_EVID_HOOK_OFF(2); TS = 2010
Line State: OHT -> Talk, Feeding Dial Tone
Received event: VP_EVID_HOOK_OFF(0); TS = 29640
Line State: OHT -> Talk, Feeding message
Line: 0, DTMF Digit: 11, Timestamp 41504: Edge: Leading

Command     arguments       End  Comments
<*>                              Print this help
<00>        <Option ID>     <*>  Get an Option
<20>        <num meter>     <*>  Start metering (Pulses specified)
Line: 0, DTMF Digit: 11, Timestamp 41824: Edge: Trailing
```