

Programmers' Guide to WV5_DLL API

Revision: 0.9

Date: May 5, 2009

Revision History

Rev. 0.9 May 5, 2009

Document corresponds with the P1R2 release of the WaveVision-5 package.

1 Introduction

The WV5 DLL provides a set of functions that will allow the caller to configure and communicate with all the WaveVision-5 family of data I/O boards and the devices connected to those I/O boards. The main objective of the DLL is to provide access to the data both generated and consumed by the device under test (DUT). As of this revision, the devices are currently limited to ADCs. The support for DACs will be provided at a future time.

Although the user of this API will only be interacting with the exported function calls of the DLL, there are several other blocks of software and hardware that are involved in providing the services of the DLL. These blocks include the low-level USB communication drivers, the firmware residing on the WaveVision capture board, the FPGA to capture the ADC output, and the ADC itself. Please refer to the WV5 System Developers' Guide for more details on these other blocks.

The beginning portion of this document should be read and understood by all those who eventually use the DLL. The reason is the beginning portion describes the behavior of the DLL which needs to be understood to effectively debug problems. The document also provides a link to an associated document that describes in detail the C interface exported by the DLL. The C technical reference details will need to be understood by those writing application programs to interface directly to wvdll.h.

Please note that this API is designed with a C/C++ application in mind that is running on a Windows platform.

1.1 WaveVision-5 System Background

Though it has several variants, collectively *WaveVision* refers to National's common evaluation platform for signal-path solutions - be they evaluation boards or more complex reference boards. The following gives a high-level overview of this platform's architecture and capabilities.

Key Features

- Runs on a Windows XP system using USB 2.0/1.0 port.
- Capable of working across a broad range of signal path boards - from precision (16-bit or greater resolution at ksamples/sec type speeds) to very high speed (up to 16-bit resolution at hundreds of Msamples/sec type speeds).
- Easy adaptability to future board designs.
- Standardized interfaces at the top and the bottom of the "software/hardware stack" that allow a user to leverage the WV5 core functionality and spend his resources developing custom GUI software and/or the future hardware designs.
- Jumper less plug-and-play experience for the user due to auto-discovery and configuration capabilities.
- Supported by the Strategic Applications group.

High-Level Architecture

Figure 1 shows the high-level architecture of the WaveVision system.

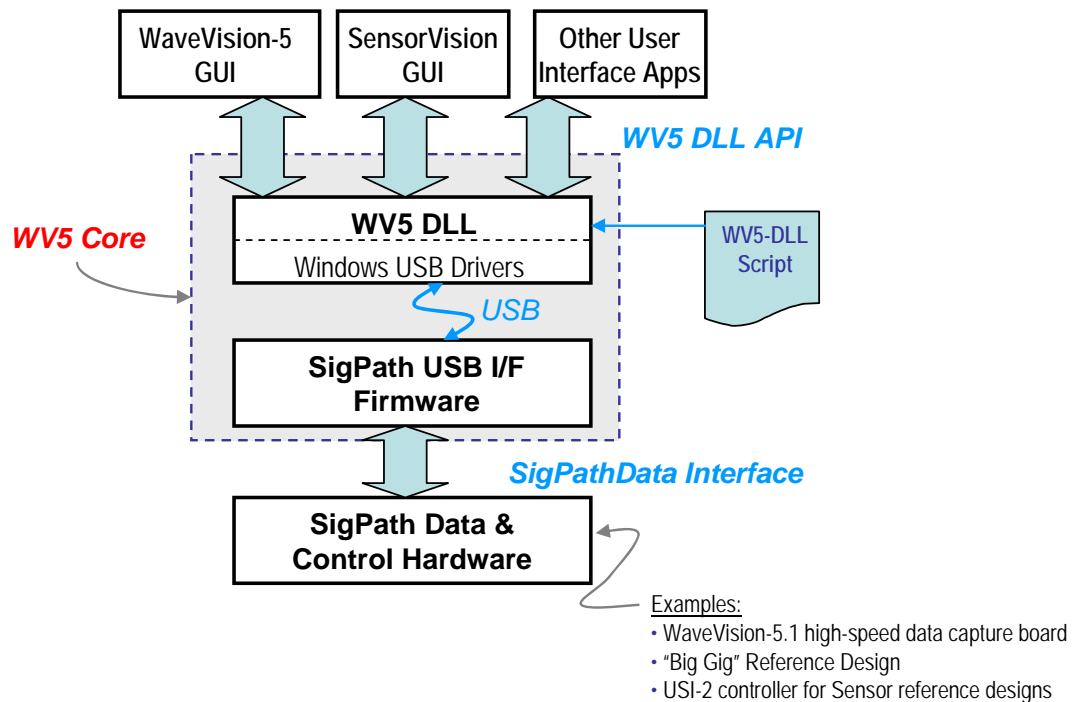


Figure 1 - High-level architecture of the WaveVision system

National has documented the top-level interface (the WV5_DLL API) into the WV5 Core software so that our internal applications engineers may develop their own application software for their needs. One may write a custom GUI, or even a command-line type of simple program quickly to accomplish a specific lab task. Or one may write a more ambitious GUI or a purely analytical software routine such as Matlab that does something that the commercial apps like WaveVision-5 or SensorVision-2008 cannot.

***** IMPORTANT NOTE: The WV5_DLL API is sometimes provided to our valued customers as courtesy to help them develop solutions using National silicon. It is provided as-is and there is no product support made available *****

1.2 *Hardware-related services provided by the DLL*

- ❑ Dynamically recognize both the capture board and DUT
- ❑ Download microcontroller firmware to capture board
- ❑ Program FPGA using appropriate image
- ❑ Read/write FPGA registers
- ❑ Read/write Cypress microcontroller registers
- ❑ Read/write DUT registers if applicable
- ❑ Access to the I2C bus on the Cypress microcontroller to allow for communication with other peripherals on the boards
- ❑ Return data deserialized by the FPGA
- ❑ Support multiple capture boards and DUTs simultaneously
- ❑ Detect USB connect and disconnect conditions
- ❑ Detect loss of clock and loss of power on the capture board and DUT

1.3 *Software features of the DLL*

- ❑ Thread-safe operation with synchronized messages
- ❑ Access to the same capture board from multiple DLL clients simultaneously. There may be situations when multiple GUIs or multiple client applications need to communicate with the same capture board. This is supported by the DLL. Typically, each application will load its own individual instance of the DLL. However, the DLL uses a system-wide resource to guarantee that only one DLL instance is accessing the hardware at any given time. Due to potential DLL version problems, the applications are advised to always use the same DLL from the hard drive.
- ❑ Callbacks to inform user of asynchronous hardware events
- ❑ Basic layout of GUI designed by the hardware engineer that best illustrates the features of the device
- ❑ Unified set of functions to communicate with DUT regardless of physical hardware mechanism

2 Environment

The WV5 Core release consists of only a few files and one large firmware directory.

2.1 Files

Name	Description
wv.dll	DLL
libcint.dll	C Scripting engine
wv5.sys	low-level Windows communication driver
wv5.inf	wv5.sys information file
wv.dll.h	Programmer's interface

2.2 hardware directory

Name	Description
firmware_images*.bix	Cypress microcontroller images
fpga_images*.bit	FPGA images
scripts\image_map.xml	File to map boards and devices to their support files
scripts*.brd.xml	Describes the hardware layout of a capture board
All other scripts*.xml	Device-specific configuration files. These files are referred to as device files.
scripts*.brd.cpp	Board-level C scripting files
scripts*.dut.cpp	DUT C scripting files
scripts\include*	C scripting include files

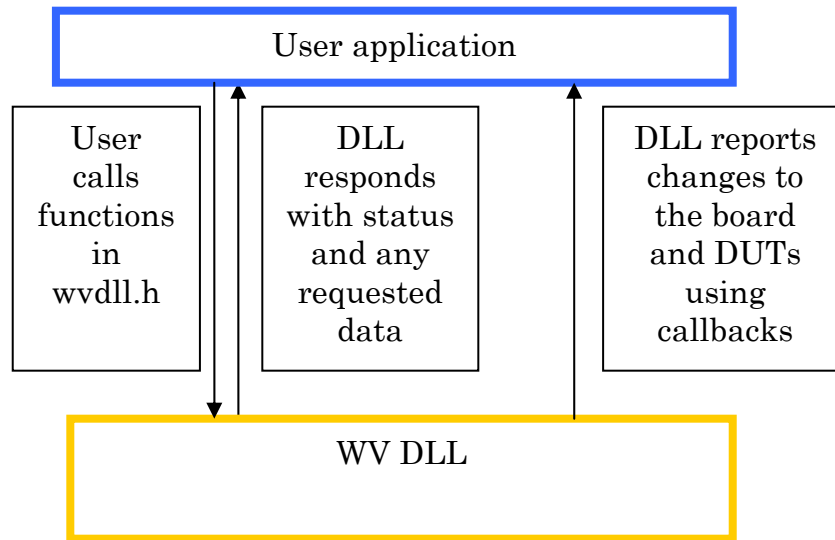
The wv5.sys and wv5.inf file should be installed using the standard procedure for installing Windows drivers.

The wv.dll file can exist anywhere. However, it is customary to have this reside in the same location as the executable the uses the DLL. Also, it is customary to have the firmware directory reside in the same level as the DLL. However, the location of the firmware directory may be modified using a DLL function call.

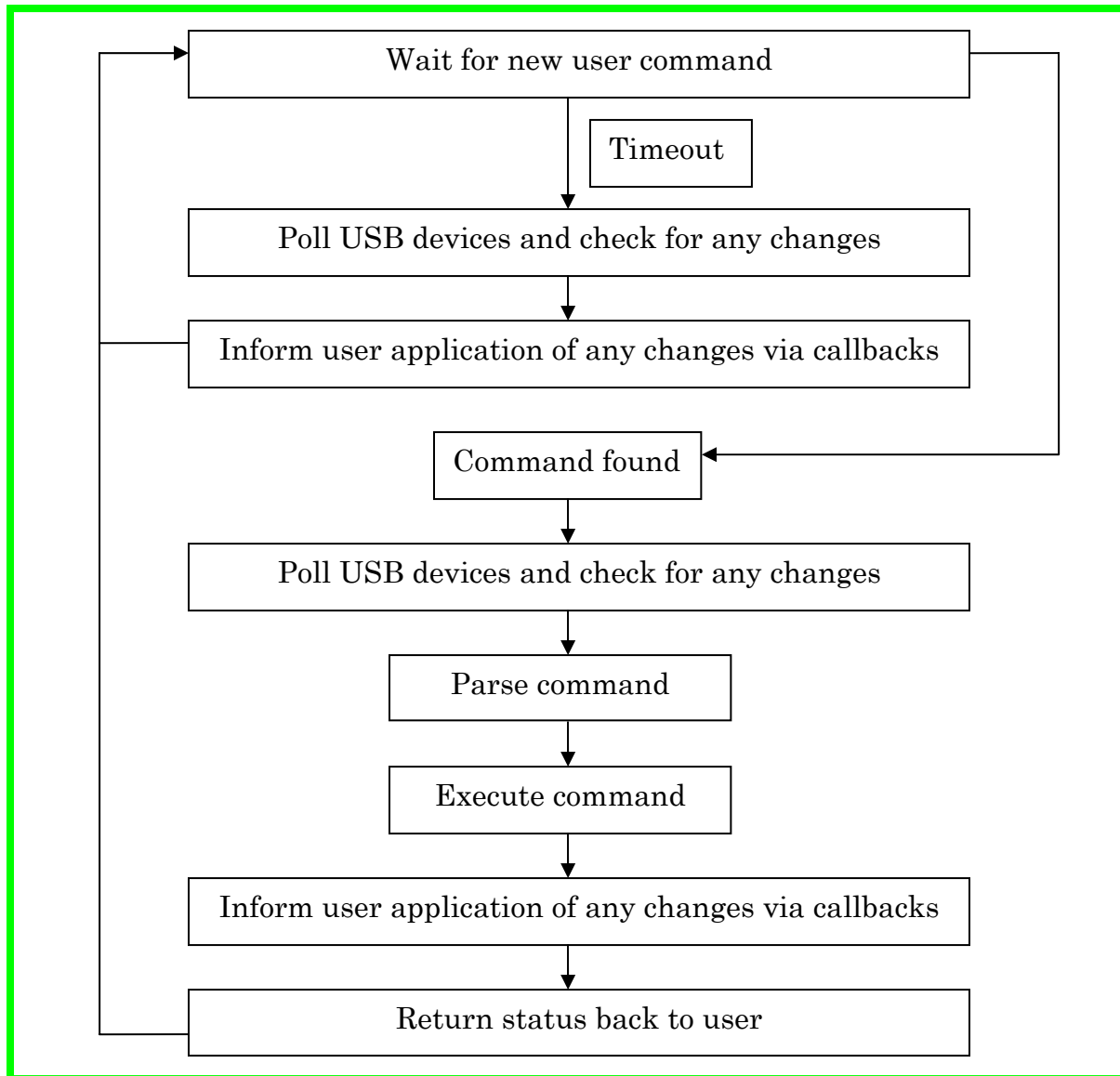
2.3 Quick note regarding device files

To support an ever increasing number of devices, all of the device-specific logic is implemented in the device files. This reduces the number of changes necessary to the DLL and the DLL API. These files are written by National Semiconductor hardware/application engineers and are not intended to be manipulated by the caller of the DLL.

3 Basic User and DLL Interaction



4 Basic State Machine in DLL



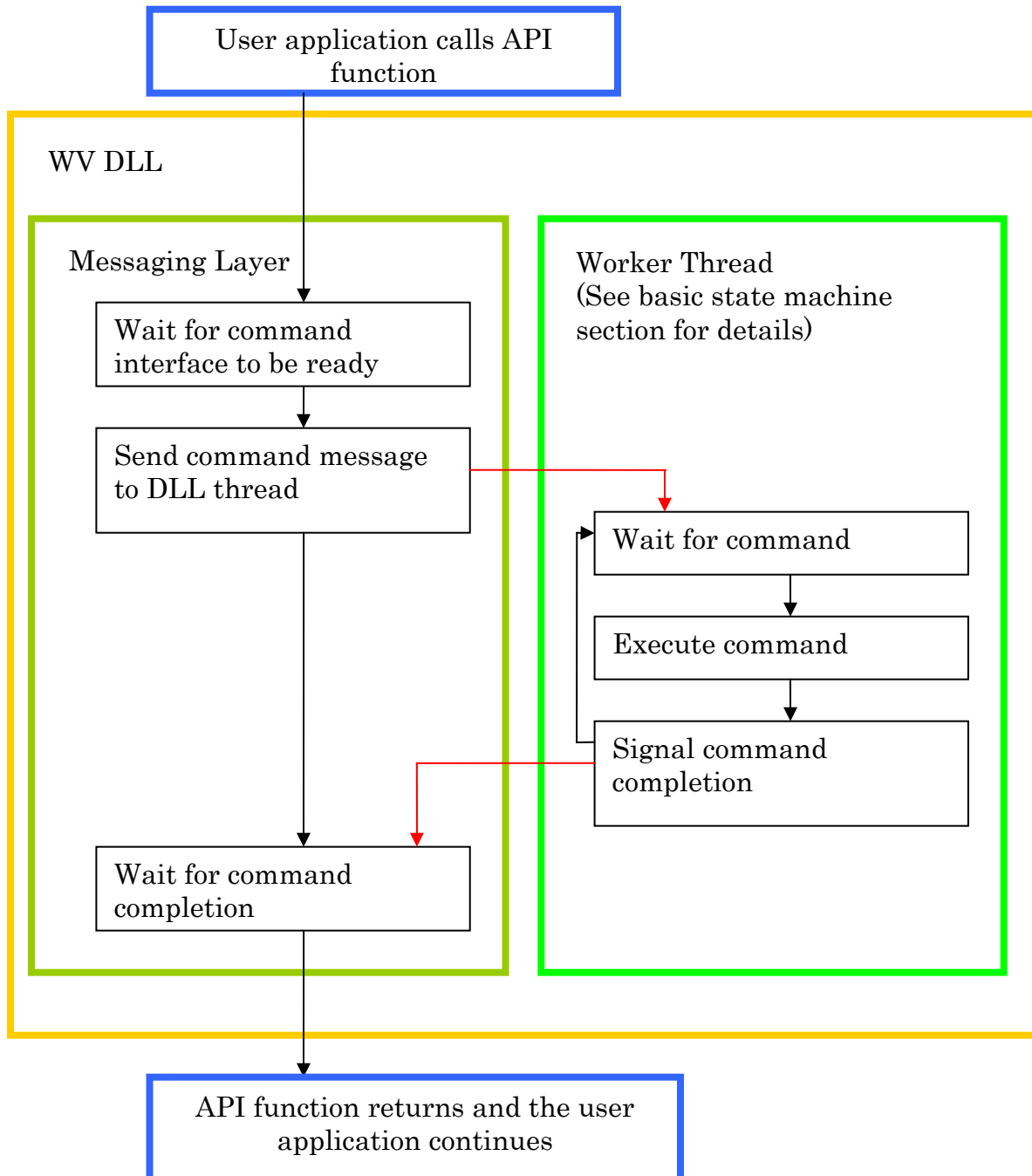
Some notes:

- ❑ The DLL is always checking for device status changes.
- ❑ If the DLL is waiting for a response from the firmware and the USB connection was interrupted, then it is possible the "execute command" stage will take a long time while waiting for the Windows USB driver to timeout.
- ❑ The DLL only executes one command at a time. This fact along with the global locking mechanism guarantees that there is only one active request to the hardware at any given time.

5 DLL Thread

Legend:

RED – flow of messages between threads/process



6 Example Code

```
int main() {
    WvWord    board_count;
    WvString* board_names;
    Wv_Version_Number version_dll;
    WvWord    code_version;
    Wv_Version_Number version_firmware;
    Wv_Version_Number version_fpga;
    // Enum DUTs
    WvWord    dut_count;
    WvString* dut_names;
    WvDUTInfo info;
    Wv_Version_Number version_dut;
    Wv_Capture_Request req;
    WvBool    data_valid;
    WvWord    captured_data[4096];

    WvDebugModuleSetOutputFilename("dlloutput.txt");

    // This is absolutely necessary to start the DLL
    WvDebugModuleModifyAll(TRUE);

    if (!WvBoardEnum(&board_count, &board_names, &version_dll)) {
        return false;
    }
    if (board_count == 0) {
        // No boards thus nothing to do.
        return -1;
    }
    if (!WvBoardOpen(0, "", &code_version, &version_firmware)) {
        // Something went wrong.
        return -1;
    }
    if (!WvBoardDUTEnum(0, &dut_count, &dut_names)) {
        // Something went wrong.
        return -1;
    }
    if (dut_count == 0) {
        // No DUTs. Nothing to do.
        return -1;
    }
    if (!WvBoardLoadFPGA(0, 0, "", &version_fpga)) {
        // Something went wrong.
        return -1;
    }
    if (!WvBoardReadDUTInfo(
        0,
        0,
        &info,
        &version_dut)) {
        // Something went wrong.
    }
```

```
        return -1;
    }
    if (info.ChannelCount == 0) {
        // No channels to capture.  Nothing to do.
        return -1;
    }
    req.DACFreq = 0;
    req.TransferSize = info.pChannelData[0].TransferMinSize;
    req.ChannelDataIndex = 0;
    req.HistogramEnable = 0;
    req.HistogramMaxBin = 0;
    req.DataFormat = info.pChannelData[0].default_data_format;

    if (!WvBoardWriteCaptureStart(0, 0, &req)) {
        // Something went wrong.
        return -1;
    }
    if (!WvBoardReadCaptureData(0, 0, captured_data, 4096,
&data_valid)) {
        // Something went wrong.
        return -1;
    }
    if (!WvBoardWriteCaptureEnd(0, 0)) {
        // Something went wrong.
        return -1;
    }
    WvBoardClose(0);
    WvShutdown();
    return 0;
}
```

7 Hardware Boot Process

This section will describe major steps taken by the DLL and firmware during the initialization process. The process described in this section assumes the following conditions:

- ❑ Board has just been power cycled
- ❑ Physical USB connection between the board and the computer has just been established
- ❑ DLL has not been launched

Any reference to a .c file in the "Firmware" blocks refers to the file that contains the implementation of the function that handles the command. mm_fpga.c is the file for the new WV5 boards. There are other boards that use different files. Please see the firmware documentation for more details relating to the USB1 firmware and the USB2 firmware the supports the legacy FPGA architectures.

DLL: recognizes board

DLL: processes Cypresss EEPROM

DLL: downloads Cypress firmware

Uses image_map.xml to determine with .bix file to download

DLL: Programming complete

Firmware: usb.c:TD_init

Setup endpoints

Firmware: mm_fpga.c:mm_fpga_init_ports

Initial setup for GPIO pins

Firmware: usb.c:TD_Poll

Main loop

Does nothing until command arrives

At this point, the firmware is now ready to accept commands from the PC

DLL: sends CMD_ID_PING

Check for alive

Firmware: mm_fpga.c:mm_fpga_parse_command_packet

Responds to CMD_ID_PING

DLL: sends CMD_ID_EEPROM_READ

Read Cypress EEPROM

Firmware: mm_fpga.c:mm_fpga_parse_command_packet

Processes CMD_ID_EEPROM_READ command

eprom_imp.c:eprom_read

Returns EEPROM data

DLL: parses board-level script file

This script file contains additional information regarding GPIO pin usage

DLL: sends CMD_ID_PIN_CONFIG

Firmware: mm_fpga.c:mm_fpga_parse_command_packet

Processes CMD_ID_PIN_CONFIG

pin_imp.c:pin_config

Processes pin configuration

DLL: sends CMD_ID_SEQ

This sequence of commands makes sure all pins configured for output properly have their corresponding output enable signals set and sets their default output state to low

Firmware: mm_fpga.c:mm_fpga_parse_command_packet

Processes CMD_ID_SEQ

cmd_seq_imp.c:cmd_seq_process

Processes the commands

DLL: sends CMD_ID_ENABLE_DUT_EEPROM_QUERY

This command tells the firmware to occasionally poll for DUT EEPROM existence. This is used for detecting hot plug out. For all boards newer than WaveVision-5.1 (eg, USI-2, WaveVision-5.2, SPIO-5.5 and later), the DUT_Presence_Detect pin is used instead of trying to detect I2C communication failure. However, for legacy reasons this functionality remains in the code.

Firmware: mm_fpga.c:mm_fpga_parse_command_packet

Processes CMD_ID_ENABLE_DUT_EEPROM_QUERY

DLL: sends CMD_ID_FIRMWARE_INFO

Firmware: mm_fpga.c:mm_fpga_parse_command_packet

Processes CMD_ID_FIRMWARE_INFO

Returns a version string

DLL: sends CMD_ID_EEPROM_READ

Read DUT EEPROM

Firmware: mm_fpga.c:mm_fpga_parse_command_packet

Processes CMD_ID_EEPROM_READ command

eeeprom_imp.c:eeeprom_read

Returns EEPROM data

DLL: sends CMD_ID_FPGA_PROGRAMMED

Determine current FPGA programmed state

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Processes CMD_ID_FPGA_PROGRAMMED command

DLL: sends CMD_ID_FPGA_APPLY_POWER
Apply power using PIN_NPOWER_EN

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Processes CMD_ID_FPGA_APPLY_POWER command

DLL: sends CMD_ID_FPGA_SET_64_SIZE
Informs firmware of FPGA image size in 64 byte chunks

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Processes CMD_ID_FPGA_SET_64_SIZE command

DLL: sends CMD_ID_FPGA_PROGRAM
Program FPGA

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Processes CMD_ID_FPGA_PROGRAM command
utils.c:common_program_fpga
Do bit-bang programming

DLL: sends CMD_ID_FPGA_CHECK_PROGRAM_DONE
Determine programming state

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Processes CMD_ID_FPGA_CHECK_PROGRAM_DONE command

DLL: sends CMD_ID_FPGA_GET_INTERFACE_ID
Determine interface id

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Processes CMD_ID_FPGA_GET_INTERFACE_ID command

DLL: sends CMD_ID_FPGA_RESET
Reset FPGA

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Processes CMD_ID_FPGA_RESET command

DLL: sends CMD_ID_PING
Check for alive

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Responds to CMD_ID_PING

DLL: sends CMD_ID_GET_FPGA_VERSION
Get version

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Responds to CMD_ID_GET_FPGA_VERSION

DLL: parses out DUT-level script file

DLL: executes all commands in the <boot_seq> segment of the script file

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Process all firmware-related commands from <boot_seq>

8 Hardware Activities While Idle

While there are no user commands being executed, the firmware and the DLL are in constant communication. This section describes the actions taking during this idle time. The main goal is to poll for changes in connectivity and power status.

Any reference to a .c file in the "Firmware" blocks refers to the file that contains the implementation of the function that handles the command. mm_fpga.c is the file for the new WV5 boards. There are other boards that use different files. Please see the firmware documentation for more details relating to the USB1 firmware and the USB2 firmware the supports the legacy FPGA architectures.

DLL: sends CMD_ID_CHECK_DUT

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Responds to CMD_ID_CHECK_DUT

DLL: sends CMD_ID_PING
Firmware will place some power status info in the response

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Responds to CMD_ID_PING

9 Hardware Activities During Capture

This section describes the commands and replies to and from the firmware during the capture process.

Any reference to a .c file in the "Firmware" blocks refers to the file that contains the implementation of the function that handles the command. mm_fpga.c is the file for the new WV5 boards. There are other boards that use different files. Please see the firmware documentation for more details relating to the USB1 firmware and the USB2 firmware the supports the legacy FPGA architectures.

DLL: sends CMD_ID_CAPTURE_FPGA_INIT
Initial configuration

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Responds to CMD_ID_CAPTURE_FPGA_INIT
mm_fpga.c:capture_fpga_init

DLL: sends CMD_ID_CAPTURE_INIT
More configuration

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Responds to CMD_ID_CAPTURE_INIT
mm_fpga.c:capture_init

DLL: sends CMD_ID_CAPTURE_START
Start the acquisition

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Responds to CMD_ID_CAPTURE_START
mm_fpga.c:capture_start

DLL: sends CMD_ID_CAPTURE_SEND_DATA
Get the data

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Responds to CMD_ID_CAPTURE_START
mm_fpga.c:capture_send_data

DLL: sends CMD_ID_PING
Make sure firmware still alive

Firmware: mm_fpga.c:mm_fpga_parse_command_packet
Responds to CMD_ID_PING

10 User Commands for Capturing

The process of capturing data requires three commands. These commands must be executed for each iteration of the capture. This means all three must be called if the user wants to repeatedly capture data.

```
WvBoardWriteCaptureStart  
WvBoardReadCaptureData  
WvBoardWriteCaptureEnd
```

11 Choosing the Support Files

This describes how the DLL chooses the files in the firmware directory. This process is automated and requires no decision making from the application. This appendix is strictly for information purposes.

(Please note that higher-level description of the WaveVision-5 system is provided in the WV5 System Developers' Guide).

Before any decision making, the DLL loads the file `firmware\image_map.xml`. This file lists any exceptions that need to be considered when determining which file to load.

When a board is connected to the USB port, the DLL identifies the specific board using the USB VID and PID numbers. These numbers are used to find the board properties entry in the image map. The most important piece of information extracted from this process is the nickname of the board. Although this nickname is not exposed to the user, it is used throughout the image map.

Once the board is identified, the Cypress firmware must be loaded. First, the CPU exception section of the image map is consulted to see if there is an exception for this board. If there is a match, then the DLL uses the value in the override field as the basename. Otherwise the DLL constructs the default basename using the form "boardnick_cputype," where "boardnick" is the board nickname and "cputype" is the CPU type from the board properties. Once the basename is identified, the wildcard name "basename*.bix" is used to look for the CPU image.

Once the CPU is programmed, the DLL attempts to configure the board. This process involves configuring how the Cypress GPIO pins are used. The information regarding the pin allocation is usually found in a board level script file. A similar process for finding the CPU file is used for the board level script file. First the image map is consulted for any board script file exceptions. If one is found, then the DLL uses the value in the override field as the basename. Otherwise the DLL constructs the default basename using the form "boardnick," where "boardnick" is the board nickname from the board properties. Once the basename is identified, the wildcard name "basename*.brd.xml" is used to look for the board level script. This script is optional and may or may not exist.

Once the board is configured, the application may attempt to enumerate all the DUTs. This is important because this process tells the DLL which DUTs are connected. The DLL uses the DUT names as a parameter into determining which FPGA image to load.

The application may then attempt to load the FPGA. The DLL first consults the image map for FPGA image exceptions. If one is found, then the DLL uses the value in the override field as the basename. Otherwise the DLL constructs the default basename using the form "boardnick_fpgatype_dut," where "boardnick" is the board nickname, "fpgatype" is the FPGA type, and "dut" is the name of the DUT. Once the basename is identified, the wildcard name "basename*.bit" is used to look for the FPGA image.

Lastly, the device-specific script file must be loaded. The DLL consults the image map for register exceptions. If one is found, then the DLL uses the value in the override field as the basename. Otherwise the DLL constructs the default basename using the form "boardnick_fpgatype_dut," where "boardnick" is the board nickname, "fpgatype" is the FPGA type, and "dut" is the name of the DUT. Once the basename is identified, the wildcard name "basename*.reg.xml" is used to look for the device file. This file is also optional.

Support file	Default basename	Wildcard
CPU image	boardnick_cputype	basename*.bix
Board level script	boardnick	basename*.brd.xml
FPGA image	boardnick_fpgatype_dut	basename*.bit
Device script	boardnick_fpgatype_dut	basename*.reg.xml

The reason for the wildcard is to allow these files to have other identifying hints. For instance, there is a file called `wv5_xc4vlx25_adc14ds105_20080220_spec_compliant.bit`. As the name hints, this image was built on February 2, 2008 and is compliant with a specification. The basename for this will be "wv5_xc4vlx25_adc14ds105" and so the wildcard "wv5_xc4vlx25_adc14ds105*.bit" will be able to find this bit file.

12 DLL_API Technical Reference

The API provides a set of functions that can be called by the Application. It also implements several structures necessary for its operation. A complete technical reference to these is provided in an associated file: [dll_help.chm](#)

In addition to the technical reference, the support package for the WV5_DLL API also includes the header and library files.